Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

dr. Nejc Carl

# Razvoj večslojnega informacijskega sistema za določanje standardne klasifikacije bolezni

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Matjaž Kukar

Ljubljana, 2017

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Raziščite, ter kvalitativno in kvantitativno ovrednotite področje uporabe mednarodne klasifikacije bolezni (ICD) v Sloveniji in po svetu. Razvijte večslojni informacijski sistem, ki bo s pomočjo sodobnih tehnologij procesiranja naravnega jezika preko spletne aplikacije omogočal preprosto in fleksibilno iskanje po mednarodni klasifikaciji bolezni. Uspešnost iskanja eksperimentalno ovrednotite.

*Mentorju, prof. dr. Matjažu Kukarju, se zahvaljujem za idejno zasnovo, za izbiro metod, za preproste odgovore na zahtevna vprašanja in za poglobljene pogovore, ki jih je pospremil z veliko mero prijaznosti in vljudnosti.*

*Staršem se zahvaljujem za pomoč in razumevanje v največji možni meri, česar si vsakdo želi, toliko bolj, kdor hodi po svoji poti.*

Posvečam mojim staršem. Vedno sta poudarjala pomembnost izobrazbe – upam da nisem pretiraval.

# Kazalo

# Seznam uporabljenih kratic

| kratica | angleško | slovensko |
|---|---|---|
| **ACID** | atomicity, consistency, isolation, durability | atomarnost, konsistentnost, izolacija, trajnost |
| **DOM** | document object model | objektni model dokumenta |
| **EJB** | Enterprise JavaBeans | javanska strežniška zrna |
| **JAX-RS** | Java API for RESTful web services | javanski programski vmesnik za spletne storitve REST |
| **JDBC** | Java database connectivity | standard za baze podatkov pod javo |
| **JPA** | Java persistence API | javanski trajnostni programski vmesnik |
| **JSON** | Javascript object notation | objektni zapis za javascript |
| **JSR** | Java specification requests | javanski zahtevek za specifikacijo |
| **ICD/MKB** | international classification of diseases | mednarodna klasifikacija bolezni |

| | | |
|---|---|---|
| **ICDS/IMKB** | International Classification of Diseases Search | Iskanje mednarodne klasifikacije bolezni |
| **NLP** | natural language processing | procesiranje naravnega jezika |
| **ODBC** | open database connectivity | odprti povezljivost podatkovnih baz |
| **ORM** | object–relational mapping | objektno–relacijska preslikava |
| **OOP** | object oriented programming | objektno usmerjeno programiranje |
| **REST** | representational state transfer | predstavitev prenosa stanja |
| **SUPB** | database management system | sistem za upravljanje s podatkovnimi bazami |
| **SSL** | transport layer security | varnost prenosnega sloja |
| **WHO** | world health organization | svetovna zdravstvena organizacija |

# Povzetek

**Naslov:** Razvoj večslojnega informacijskega sistema za določanje standardne klasifikacije bolezni

Diplomsko delo obravnava pomemben del izmenjave zdravstvenih informacij – Mednarodno klasifikacijo bolezni. Namen dela je olajšati problem identifikacije kategorije, s katero bomo klasificirali zdravstveno informacijo. Problem, s katerim se srečujejo zdravstveni delavci je, da Mednarodna klasifikacija bolezni vsebuje skoraj 19000 vnosov, ki so enolično določeni s kategorijo in dvema podkategorijama. Zdravstveni delavci imajo običajno v spominu samo najbolj pogosto uporabljane kategorije. Namen te aplikacije je, da uporabnik vpiše nekaj ključnih besed, nato pa pridobi relevantne vnose Mednarodne klasifikacije bolezni. V ta namen smo implementirali rešitev v obliki spletne aplikacije, pri kateri smo za izgradnjo odjemalca uporabili ogrodje Angular, na strežniku smo uporabili poslovno izdajo Jave, za podatkovno bazo pa smo uporabili MySQL. Z uporabo tehnik procesiranja naravnega jezika, predvsem lematizacije, smo ublažili vplive morfološke dinamičnosti slovenskega jezika. Za izboljšanje učinkovitosti iskanja po opisih Mednarodnih klasifikacij bolezni pa smo si pomagali še z MySQL-ovim polno-tekstovnim iskanjem. Implementacija se je pokazala kot uspešna, saj s preprostim vpisom nekaj ključnih besed omogoča inkrementalno iskanje kandidatov v realnem času.

**Ključne besede:** MKB, MKB-10, Mednarodna klasifikacija bolezni, LemmaGen, Java EE, MySQL, Angular.

# Abstract

**Title:** Development of Multi-Tier Information System for Determination of International Statistical Classification of Diseases and Related Health Problems

The subject of this diploma thesis is exchange of medical information with International Statistical Classification of Diseases and Related Health Problems. The purpose of this work is to facilitate the classification of medical information according to the aforementioned standard. International Classification of Diseases is composed of almost 19000 entries that are uniquely defined by category and two subcategories. Health workers have usually memorized only the most frequently used categories, so they often have to find the appropriate coding for medical information. The aim of this application is to respond to keywords entered by the user with the relevant ICD entries. To this end, we have implemented a web application, that we build using the framework Angular on the client, the Java EE framework on the server and MySQL as a RDBMS. To dampen the effects of morphological dynamism of the Slovene language we used natural language processing techniques, primarily lemmatisation. Additionally, MySQL full-text search was used to help us find the appropriate classification. Implemented application allows us to determine relevant ICD entries in real time, with only a few keywords as input. Thus, we consider the implementation successful.

**Keywords:** ICD, ICD-10, International Classification of Diseases, LemmaGen, Java EE, MySQL, Angular.

# Poglavje 1

# Uvod

## 1.1 Motivacija za izdelavo naloge

Po vodilih svetovne zdravstvene organizacije (WHO) [5] se kot standardne oznake bolezni z namenom primerljivosti, statistike in zavarovalniških potreb uporabljajo šifranti ICD-10 (International Classification of Diseases) oziroma slovenski MKB-10 (Mednarodna klasifikacija bolezni) [15]. Slovenska izdaja MKB-10 po vsebini ustreza šesti izdaji izvirnika ICD-10-AM (avstralska modifikacija) [7]. Zdravniki in drugi zdravstveni delavci so pri svojem delu pogosto obvezani uporabiti mednarodno šifro za bolezen. Problem, s katerim se pri tem srečujejo, je, da Mednarodna klasifikacija bolezni vsebuje skoraj 19000 vnosov, ki so enolično določeni s kategorijo in dvema podkategorijama. Zdravstveni delavci imajo običajno v spominu samo najbolj pogosto uporabljane kategorije. Za identifikacijo ustrezne šifre bolezni bi jim lahko pomagala aplikacija, ki bi iz ključnih besed opisa bolezni v naravnem jeziku ponudila nabor šifer bolezni.

Koristno bi bilo, če bi obstajala aplikacija, v katero bi lahko vnesli opis bolezni v naravnem jeziku in s pomočjo take aplikacije identificirali ustrezno šifro bolezni.

## 1.2 Hipoteze

Diplomsko delo bo utemeljeno na sledečih delovnih hipotezah:

- možno je implementirati fleksibilno iskanje v naravnem jeziku po opisih bolezni, ki se izvaja nad standardnimi klasifikacijami bolezni
- iskanje po opisih je možno integrirati v spletno rešitev, kjer se na podlagi ustrezno obdelanega vnosa ključnih besed prikažejo klasifikacije bolezni, ki najbolj ustrezajo vnešenemu opisu.

## 1.3 Cilj diplomske naloge

V okviru diplomske naloge bomo razvili programsko opremo za iskanje mednarodne klasifikacije bolezni (MKB). Končni izdelek diplomske naloge bo aplikacija, v kateri v iskalno polje vpišemo fragment opisa bolezni v naravnem jeziku (stavke, ključne besede, fraze), ki ga bo aplikacija poslala na strežnik, kjer se bo vsaka od poslanih besed lematizirala, in kjer se bodo poiskali MKB-ji, ki te leme vsebujejo. Ujemanje lem in MKB-jev se bo točkovalo. Odjemalec bo prejel urejen seznam MKB-jev, najprej tistih z najboljšim ujemanjem. Prikazane MKB-je bo možno filtrirati po besedah ali kodah, ki jih vsebuje MKB.

# Poglavje 2

# Mednarodna klasifikacija bolezni

Mednarodna klasifikacija bolezni (MKB) je alfanumerični kodirni sistem za zapis bolezni in zunanjih vzrokov poškodb. Razvita je bila s pomočjo zdravnikov in zdravstvenih administratorjev. Organizirana je po delu telesa in po etiologiji, kode pa so dolge tri, štiri ali pet znakov. V Sloveniji uporabljamo ICD-10-AM, to je deseto izdajo MKB, prirejeno za Avstralijo in modificirano pri nas (šesta izdaja).

## 2.1 Zgodovina

Predhodnika Mednarodne klasifikacije bolezni je leta 1893 prvič objavil Mednarodni inštitut za statistiko pod imenom Mednarodni seznam vzrokov smrti. [5] Mednarodna zdravstvena organizacija (WHO) je leta 1948 objavila šesto verzijo mednarodne klasifikacije bolezni, ki so ji bili dodani tudi vzroki smrti. Nomenklaturna pravila, ki jih je WHO sprejel leta 1967, narekujejo, da njegove države članice za beleženje statistike vzrokov smrti uporabljajo zadnjo revizijo mednarodne klasifikacije bolezni (MKB). MKB je bil večkrat posodobljen v skladu z napredkom v medicini. Zadnja – deseta – izdaja je bila sprejeta leta 1990 na 43. zasedanju WHO-ja. MKB uporabljajo v več kot

sto državah sveta. Citiran je v več kot 20.000 znanstvenih člankih.

## 2.2 Namen in uporabe

MKB je temelj za določanje trendov in zdravstvenih statistik. Prav tako je mednarodni standard za poročanje o boleznih in zdravstvenih stanjih. MKB se uporablja v klinične in raziskovalne namene kot diagnostični klasifikacijski standard. MKB hierarhično razvršča vse bolezni, motnje, poškodbe in povezana zdravstvena stanja. V svetovnem merilu je MKB uporabljan za dodelitev približno 70% zdravstvenih izdatkov. [6] Mednarodna klasifikacija bolezni je bila prevedena v 43 jezikov. MKB omogoča:

- preprosto hranjenje, dostop do in analizo podatkov
- izmenjavo in primerjavo zdravstvenih informacij med bolnicami, regijami, okolji in državami
- primerjavo podatkov v nekem okolju skozi čas
- nadzor incidence in prevalence bolezni, nadzor nad z zdravljenjem povezanimi izplačili in trendi alokacije finančnih sredstev
- štetje smrti, bolezni, poškodb, simptomov in razlogov za sprejem, dejavnikov, ki vplivajo na zdravje in zunanjih vzrokov bolezni

# Poglavje 3

# Uporabljene tehnologije

## 3.1 Strežniški del aplikacije - Java EE

Poslovna izdaja Jave (angl. Java Enterprise Edition) je specifikacija programskega vmesnika, ki zajema mnoge tehnologije kot so javanska strežniška zrna (EJB) in javanski standard za REST spletne storitve [40]. V nadaljevanju so opisane nekatere v tem diplomskem delu uporabljene javanske tehnologije.

### 3.1.1 JAX-RS - javanski standard za REST spletne storitve

#### REST spletne storitve

REST je kratica za Representational State Transfer, ki je uveljavljena arhitektura aplikacij, kjer komunikacija med odjemalcem in strežnikom poteka tako, da so viri na strani strežnika predstavljeni s spletnimi storitvami, do katerih dostopamo. [12] Klic programskega vmesnika je tradicionalno predstavljal klic metode na daljavo, REST arhitekura aplikacij pa smatra podatke in funkcionalnost kot vire, do katerih dostopamo preko URL-ja.

Za REST arhitekturni slog so značilni naslednji principi:

- **Identifikacija virov preko URL-ja:** Do virov REST spletnih stori-

tev lahko dostopamo preko spletnih naslovov.

- **Enotni vmesnik:** Vire upravljamo s štirimi jasno definiranimi operacijami - branje, pisanje, posodobitev in izbris. Te štiri operacije imajo svoj ekvivalent v HTTP metodah GET, POST, PUT, PATCH in DELETE.

- **Razumljiva sporočila:** Način predstavitve vira je ločen od vsebine vira tako, da lahko do istega vira dostopamo v različnih formatih kot so HTML, XML, tekst, PDF, JPEG, JSON in drugi formati.

### Razvoj spletnih storitev z JAX-RS

JAX-RS je javanski API, zasnovan za preprost razvoj aplikacij, ki uporabljajo REST arhitekturo. [12] Bolj enostaven razvoj omogoča z uporabo javanskih anotacij, s katerimi se dekorirajo razredne datoteke in tako določijo viri in operacije, ki se jih lahko izvaja nad viri. JAX-RS uporablja med drugim naslednje anotacije:

- **@Path:** S @Path anotacijo označujemo relativno URL pot, s čimer kažemo, kje se nahaja ustrezen javanski razred. Ta anotacija podpira tudi vdelovanje spremenljivk v URL.

- **@GET:** Z @GET anotacijo označimo javansko metodo, ki se bo odzivala na HTTP metodo GET. Tako je odziv vira odvisen od uporabljene HTTP metode.

- **@POST:** S @POST anotacijo označimo javansko metodo, ki se bo odzivala na HTTP metodo POST. Tako je odziv vira odvisen od uporabljene HTTP metode.

- **@PUT:** S @PUT anotacijo označimo javansko metodo, ki se bo odzivala na HTTP metodo PUT. Tako je odziv vira odvisen od uporabljene HTTP metode.

- **@DELETE:** Z @DELETE anotacijo označimo javansko metodo, ki se bo odzivala na HTTP metodo DELETE. Tako je odziv vira odvisen od uporabljene HTTP metode.
- **@HEAD:** S @HEAD anotacijo označimo javansko metodo, ki se bo odzivala na HTTP metodo HEAD. Tako je odziv vira odvisen od uporabljene HTTP metode.
- **@PathParam:** S @PathParam anotacijo označimo parameter, ki ga izvlečemo iz URL-ja.
- **@QueryParam** S @QueryParam anotacijo označimo parameter, ki ga izvlečemo iz poizvedbenih (angl. query) parametrov URL-ja.
- **@Consumes** S @Consumes anotacijo določimo MIME tip, ki ga je poslal odjemalec in ga bo použila metoda ali razred označen s to anotacijo.
- **@Produces** S @Produces anotacijo določimo MIME tip v katerem označena metoda pošlje odgovor odjemalcu.
- **@ApplicationPath** Z @ApplicationPath anotacijo določimo URL naslov aplikacije. Z @ApplicationPath določimo osnovno pot na katero se relativno sklicujejo @PATH anotacije.

### 3.1.2 EJB - javanska strežniška zrna

Javansko strežniško zrno je v programskem jeziku Java napisana komponenta, ki na strežniku ovija (angl. encapsulate) aplikacijsko poslovno logiko. [12] Poslovna logika je koda, ki izvaja namen in smisel aplikacije. V aplikaciji IMKB, ki je tema te diplome, je poslovna logika tisti del kode, ki kot argument dobi nek tekst in na podlagi tega teksta vrne ustrezne klasifikacijske kode.

Javanska strežniška zrna poenostavijo razvoj velikih porazdeljenih aplikacij. Eden od razlogov za poenostavitev je, da EJB vsebnik (angl. container) nudi sistemske storitve poslovnim zrnom in tako omogoči razvijalcu, da se

osredotoči na poslovno logiko. Upravljanje s storitvami (kot so upravljanje s transakcijami, avtorizacija in avtentikacija) vrši EJB vsebnik.

Druga poenostavitev, ki jo predstavljajo javanska strežniška zrna je prenosljivost. Komponente, ki so bile narejene za neko aplikacijo, lahko ponovno uporabimo pri neki drugi aplikaciji. V primeru, ko komponente uporabljajo standardni API, lahko tečejo na katerem koli Java EE kompatibilnem serverju.

### Kdaj uporabiti javanska strežniška zrna

Javanska strežniška zrna uporabimo, kadar želimo, da naša aplikacija izpolnjuje katero od sledečih zahtev:

- **Skalabilnost** - Ko število uporabnikov naše aplikacije preseže določeno mejo, je potrebno aplikacijske komponente namestiti na več računalnikov. Javanska strežniška zrna lahko namestimo na več računalnikov, njihova lokacija pa ne vpliva na kodo na strani odjemalca.
- **Podpora transakcijam** - S transakcijami zagotavljamo celovitost podatkov. Javanska strežniška zrna podpirajo transakcije, s katerimi upravljajo sočasni dostop do objektov v skupni uporabi.
- **Različni odjemalci** - Javanska strežniška zrna zagotavljajo, da lahko različni odjemalci enostavno dostopajo do strežniških zrn.

### Tipi javanskih strežniških zrn

Poznamo tri vrste javanskih strežniških zrn: sejna zrna s stanjem (angl. stateful session beans), sejna zrna brez stanja (angl. stateless session beans) in edinec sejna zrna (angl. singleton session bean).

### Sejna zrna s stanjem

Vrednosti spremenljivk sejnega zrna predstavljajo njegovo stanje. V sejnem zrnu s stanjem so vrednosti spremenljivk posledica interakcije med

strežnikom in odjemalcem. Ker je interakcija sestavljena iz več zahtev in odgovorov, pravimo stanju sejnega zrna pogovorno stanje. Do zrna s stanjem lahko naenkrat dostopa le en odjemalec. Dokler traja pogovor odjemalca in zrna, se stanje zrna ohranja, ko se seja konča se zaključi tudi zrno.

### Sejna zrna brez stanja

Sejna zrna brez stanja ne vzdržujejo pogovora z odjemalcem. Ko odjemalec pozove sejno zrno brez stanja, lahko spremenljivke zrna spremenijo svoje vrednosti, vendar samo za trajanje poziva. Ko pozvana metoda zaključi svoje izvajanje, se stanje zrna ne ohrani. Razen med pozivom metode so vse instance v bazenu zrn ekvivalentne, zato lahko EJB kontainer priredi posamezno zrno kateremu koli odjemalcu.

Sejna zrna brez stanja so bolj skalabilna od sejnih zrn s stanjem, saj lahko neko zrno v enem klicu priredimo nekemu odjemalcu, v drugem klicu pa priredimo to isto zrno drugemu odjemalcu. Tako aplikacije potrebujejo manj sejnih zrn brez stanja kot bi potrebovale sejnih zrn s stanjem.

### Edinec sejna zrna

Sejno zrno, ki se lahko pojavi samo kot en primerek na aplikacijo, je edinec (primer kode je na sliki 3.1). Edinec sejna zrna uporabljamo, kadar želimo, da si različni odjemalci delijo en in točno en primerek sejnega zrna.

Edinec sejna zrna imajo podobno funkcionalnost kot sejna zrna brez stanja. Razlikujejo se po tem, da sejna zrna brez stanja obstajajo v bazenu stanj, iz katerega se dodeljujejo prosta sejna zrna odjemalcem, medtem ko je edinec sejno zrno eno samo za celotno aplikacijo. Edinec sejna zrna v primeru, da je prišlo do sesutja ali zaustavitve strežnika, ne zadržijo stanja.

```
@Singleton
public class Semafor {


  private boolean gas;

  @PostConstruct
  private void init(){
    this.gas = true;
  }


  public boolean isGas() {
    return gas;
  }

  public void setGas(boolean gas) {
    this.gas = gas;
  }
}
```

Slika 3.1: Sejno zrno edinec, ki smo ga uporabili kot semafor za nadzor nad delovanjem sejnih zrn brez stanja.

### 3.1.3 WildFly - odprtokodni aplikacijski strežnik za poslovne javanske aplikacije

Eden od aplikacijskih strežnikov za Javo EE je aplikacijski strežnik JBoss. [42] JBoss je napisan v Javi zato ga lahko poganjamo na vseh operacijskih sistemih v 32 ali 64 bitni verziji. Podjetje, ki razvija aplikacijski strežnik JBoss, nudi uradno podprto, plačljivo verzijo strežnika in brezplačno verzijo. Da bi se izognili zmedi, so brezplačno verzijo poimenovali WildFly. [31]

Zagon strežnika WildFly je visoko optimiziran. Da eni storitvi ni potrebno čakati na zagon druge storitve, se vse storitve zaženejo hkrati. Vse storitve, ki niso nujno potrebne ob zagonu strežnika, ostanejo v pripravljenosti do trenutka, ko jih potrebujemo. Glavne značilnosti aplikacijskega strežnika WildFly so povezljivost, odzivnost in skalabilnost. Za izboljšanje teh značilnosti je zaslužen predvsem nov spletni strežnik - Undertow. [24] Vse WildFlyeve storitve se naložijo modularno, s čimer se prepreči nalaganje podvojenih razredov, hkrati pa omogoči delovanje strežnika na manj zmogljivih napravah.

WildFly podpira mnogo različnih funkcionalnosti in Java EE 7 standardov [30]:

- Vmesnik za nameščanje aplikacij
- Strežniška javanska zrna
- Integracija ORM orodja Hibernate [4]
- Javanske imenske in imeniške storitve (angl. Java Naming and Directory Interface (JNDI))
- Javanski transakcijski vmesnik (Java Transaction API (JTA))
- Javanska standardna povezljivost za baze podatkov (angl. Java Database Connectivity (JDBC))
- Uravnoteževanje obremenitve (angl. Load balancing)
- Vmesnik za upravljanje s strežnikom
- OSGi (Open Services Gateway initiative) ogrodje za dinamično modularnost storitev

## 3.2 Aplikacija na strani odjemalca - Angular

Angular je strukturno ogrodje za dinamične, neostranjene spletne aplikacije. [2] Z njegovo uporabo lahko HTML predloge razširimo z Angular sintakso in tako jedrnato zapišemo dele naše aplikacije. Poenostavitve, ki jih nudi Angular, so med drugim vezanje podatkov (angl. data-binding) in injiciranje odvisnosti. Angular teče na strani odjemalca, kar razbremeni strežnik.

HTML je bil zasnovan za prikaz statičnih strani in ni najbolj primeren za izdelavo dinamičnih aplikacij. Prepad med prikazom statičnih dokumentov in izdelavo dinamičnih aplikacij se običajno premošča z uporabo:

- **knjižnice** - knjižnica je zbirka funkcij, ki jih uporabljamo pri razvoju spletnih aplikacij. Med izvajanjem svojega programa po potrebi kličemo funkcije iz knjižnice. Primer knjižnice je jQuery. [13]

- **spletnega ogrodja** - spletno ogrodje je splošni primer spletne aplikacije, za katero moramo dopolniti podrobnosti. Med izvajanjem spletnega ogrodja se po potrebi kliče naša koda. Primer spletnega ogrodja sta React [21] in Angular [1].

Angularjev pristop je zmanjšanje prepada med dokumentno usmerjenim HTML-jem in potrebami aplikacije. Angular nam omogoča razširitev HTML sintakse z uporabo direktiv kot so:

- vezanje podatkov - to je sinhronizacija podatkov med HTML delom aplikacije in JavaScript delom aplikacije
- kontrolne strukture za ponavljanje, prikaz in skritje fragmentov DOM-a
- podpora obrazcem in validaciji obrazcev
- dodajanje nove funkcionalnosti elementom DOM-a
- ponovna uporaba HTML komponent

Ideja Angularja je, da je deklarativno programiranje bolj primerno za gradnjo uporabniških vmesnikov in povezovanja programja od imperativnega programiranja, ki je bolj primerno za gradnjo poslovne logike.

### 3.2.1 Prednosti uporabe ogrodja Angular

Uporaba strukturnega ogrodja Angular prinaša naslednje prednosti [2]:

- **modularnost aplikacije:** Angular aplikacija je sestavljena iz komponent, ki jih lahko ponovno uporabimo (primer komponente je na sliki 3.2). Komponenta je lahko sestavljena iz drugih komponent, v katere lahko parametre vnaša ali pa jih iz njih dobiva.

```
import { Pipe, PipeTransform } from '@angular/core';
import {Mkb} from "./mkb";

@Pipe({
  name: 'isciPoMkbju'
})
export class IsciPoMkbjuPipe implements PipeTransform {

  transform(mkbs: Mkb[], iskalnaFraza: string): any {
    let iskalneBesede: string[] = iskalnaFraza.split(/[ ]+/);
    if (mkbs.length === 0 || iskalnaFraza == "") {
      return mkbs;
    }
    let rezultat = [];
    let mkb:Mkb;
    for ( mkb of mkbs) {
      let iskalnaBeseda:string;
      let dodajMkb:boolean = false;
      for(iskalnaBeseda of iskalneBesede) {
        let regExp:string = '^.*' + iskalnaBeseda + '.*$';
        if (mkb.slovenskiNaziv.search(   new RegExp( iskalnaBeseda , 'i'))!== -1||
            mkb.id.kategorija.search(    new RegExp( iskalnaBeseda , 'i'))!== -1||
            mkb.id.podkategorija4.search(new RegExp( iskalnaBeseda , 'i'))!== -1||
            mkb.id.podkategorija5.search(new RegExp( iskalnaBeseda , 'i'))!== -1)
        {
          dodajMkb = true;
        }
        else {
          dodajMkb = false;
          break;
        }
      }
      if(dodajMkb){
        rezultat.push(mkb);
      }
    }
    return rezultat;
  }

}
```

Slika 3.2: Angular komponenta za programsko manipulacijo DOM-a. V polju pustimo samo tiste MKB-je, ki vsebujejo ustrezni niz.

- **injeciranje odvisnosti (angl. dependency injection):** Angular nam omogoča injeciranje odvisnosti v komponente. To so lahko vnaprej pripravljeni moduli kot je npr. modul HTTP, ki omogoča klicanje metod kot sta GET in POST. Injeciramo lahko tudi lastne module.
- **prijava povratnih klicev (angl. registering callbacks):** pisanje prijav povratnih klicev je dodatna koda, ki naredi program manj pregleden, Angular nam omogoča, da v samo HTML kodo dopišemo funkcije, ki se bodo ob določenih dogodkih izvedle.
- **programska manipulacija HTML-jevega DOM-a:** manipulira-

nje s HTML-jevim DOM-om je okoren in napakam podvržen temelj asinhronih aplikacij. Z deklarativnim opisom sprememb uporabniškega vmesnika nam ni treba podrobno opisovati vsake spremembe DOM-a. DOM je tudi v Angularju možno spreminjati programsko (primer na sliki 3.2, kljub temu pa pri večini aplikacij to ni potrebno.

- **pozivanje podatkov (angl. marshalling) iz in v uporabniški vmesnik:** osnovne operacije beri, piši, posodobi in zbriši predstavljajo večino operacij asinhronih aplikacij. Tok pozivanja podatkov teče od strežnika do objekta v odjemalčevi kodi. Od tu pa do HTML obrazca, kjer lahko uporabnik podatke spreminja. Odjemalčeva koda mora vnešene podatke validirati ali pa jih posreduje strežniku. Tak tok podatkov prinaša s sabo ogromno rutinskega programiranja. Angular nam ponuja mnogo funkcionalnosti, ki bi jih sicer morali sprogramirati sami in bi predstavljale rutinsko delo.

- **manj inicializacijske kode:** pri običajni asinhroni aplikaciji je potrebno veliko dela že za to, da dobimo osnovno aplikacijo tipa ,,Pozdravljen svet!“. V Angularju lahko mnogo odvisnosti injiciramo - tako lahko zelo hitro postavimo najbolj pogosto uporabljane funkcionalnosti aplikacij.

## 3.3 Trajnost aplikacijskih podatkov - MySQL

Skoraj vse aplikacije na nek način shranjujejo podatke. Trajnost podatkov je ena od osnovnih zahtev pri razvoju aplikacij – sistem, ki ne shranjuje podatkov potem ko ga ugasnemo ali potem, ko zmanjka elektrike, ne bo posebno uporaben. Trajnost podatkov pomeni, da posamezni podatki, v objektno usmerjenem programiranju so to objekti, ohranijo stanje dlje časa, kot stanje ohranja aplikacijski proces. Ko govorimo o trajnosti podatkov običajno govorimo o shranjevanju stanja objektov v relacijsko podatkovno bazo. [32]

### 3.3.1 Relacijske podatkovne baze

Sistemi za upravljanje s podatkovnimi bazami (SUPB) delujejo preko programskih vmesnikov in temeljijo na strukturiranem povpraševalnem jeziku za delo s podatkovnimi bazami (SQL). Eden od pomembnih razlogov za veliko popularnost relacijske tehnologije je njena uveljavljenost, kar je že samo po sebi razlog, da je široko sprejeta med poslovnimi uporabniki. Relacijske podatkovne baze so razširjene tudi zaradi velike fleksibilnosti in robustnosti takega pristopa upravljanja s podatki. Teorija relacijskih podatkovnih baz je že skoraj pol stoletja uveljavljena tudi v akademskih krogih, [33] zanimanje zanjo pa ne pojema. [34] Relacijske podatkovne baze niso pisane za specifičen programski jezik, prav tako ni podatkovna baza narejena za specifčno aplikacijo. Temu principu pravimo *podatkovna neodvisnost* in je ključnega pomena, da lahko *ohranjamo podatke dlje kot aplikacijsko stanje*. S pomočjo relacijske tehnologije lahko delimo podatke med različnimi aplikacijami ali med različnimi deli iste aplikacije, npr. če ima aplikacija neko funkcionalnost za zajem podatkov in če ima neko drugo funkcionalnost za njihov izpis. Neko relacijsko tehnologijo si tako lahko delijo različna računalniška okolja [32].

### 3.3.2 MySQL

MySQL je odprtokodni, večnitni sistem za upravljanje s podatkovnimi bazami (SUPB). Na voljo je pod GNU General Public License (GPL). [35] [36] MySQL je eden od najbolj uporabljanih SUPB-jev. Razlogi za to so cena (za večino primerov uporabe je zastonj), zanesljivost, zmogljivost, nabor funkcionalnosti in skalabilnost (upravlja lahko z deset tisoči tabel in milijardami vrstic podatkov). Na sliki 3.3 je prikazano generiranje podatkovne baze za našo aplikacijo, na sliki 3.4 pa generiranje tujih ključev in indeksov.

V nabor funkcionalnosti MySQL-a verzije 5.6 med drugim sodi [27]:

- široka podpora standarda SQL:1999
- lahko teče na platformah Linux, Windows, Solaris, OS X in FreeBSD

```
create table Lema
(
   lema                 char(60) not null,
   jezik                smallint not null,
   primary key (lema, jezik)
);
create table MKB
(
   kategorija           char(3) not null,
   podkategorija4       char(1) not null,
   podkategorija5       char(1) not null,
   sklop                char(7) not null,
   Skl_sklop            char(7),
   Skl_sklop2           char(7),
   slovenski_naziv      char(200),
   slovenski_naziv_kratek char(60),
   angleski_naziv       char(200),
   angleski_naziv_kratek char(60),
   krizec_etiologija    char(1),
   zvezdica_manifestacija smallint,
   veljavnost           smallint,
   skupina_obolevnosti  char(3),
   spol                 smallint,
   spol_tip             smallint,
   starost_min          smallint,
   starost_max          smallint,
   starost_tip          smallint,
   redka_diagnoza       char(1),
   morfologija          smallint,
   nespremjemljiva_glavna_diagnoza smallint,
   prijava_nalezljive_bolezni smallint,
   primary key (kategorija, podkategorija4, podkategorija5)
);
create table MKB_ima_lemo
(
   kategorija           char(3) not null,
   podkategorija4       char(1) not null,
   podkategorija5       char(1) not null,
   lema                 char(60) not null,
   jezik                smallint not null,
   primary key (kategorija, podkategorija4, podkategorija5, lema, jezik)
);
create table Poglavje
(
   poglavje             smallint not null,
   slovenski_naziv      char(200),
   angleski_naziv       char(200),
   primary key (poglavje)
);
create table Sklop
(
   sklop                char(7) not null,
   poglavje             smallint not null,
   slovenski_naziv      char(200),
   angleski_naziv       char(200),
   primary key (sklop)
);
```

Slika 3.3: Generiranje SQL tabel Lema, MKB, MKB_ima_lemo, Poglavje in Sklop. V teh tabelah so trajno shranjeni podatki o MKB-jih in pripadajoči metapodatki.

```
alter table MKB add constraint FK_Sklop1 foreign key (sklop)
      references Sklop (sklop) on delete restrict on update restrict;

alter table MKB add constraint FK_Sklop2 foreign key (Skl_sklop2)
      references Sklop (sklop) on delete restrict on update restrict;

alter table MKB add constraint FK_Sklop3 foreign key (Skl_sklop)
      references Sklop (sklop) on delete restrict on update restrict;

alter table MKB_ima_lemo add constraint FK_MKB_ima_lemo foreign key (kategorija, podkategorija4, podkategorija5)
      references MKB (kategorija, podkategorija4, podkategorija5) on delete restrict on update restrict;

alter table MKB_ima_lemo add constraint FK_MKB_ima_lemo2 foreign key (lema, jezik)
      references Lema (lema, jezik) on delete restrict on update restrict;

alter table Sklop add constraint FK_Poglavje_ima_sklop foreign key (poglavje)
      references Poglavje (poglavje) on delete restrict on update restrict;

alter table MKB add column concat_lemas char(255);

create fulltext index idxOpisiAnglescina on MKB (angleski_naziv);

create fulltext index idxOpisiConcatLemas on MKB (slovenski_naziv, concat_lemas);

create fulltext index idxOpisi on MKB (slovenski_naziv);
```

Slika 3.4: Generiranje tujih ključev in indeksov. Tabeli MKB je dodan stolpec za hranjenje lem.

- bazni programi (angl. stored procedures)
- sprožilci (angl. triggers)
- kurzorji
- pogledi
- jezik za opis podatkov (DDL)
- informacijska shema
- transakcije in skladnost z ACID načeli
- podpora za SSL
- predpomnjenje poizvedb
- gnezdena povpraševanja SELECT
- vgrajena podpora za podvojevanje
- iskanje in indeksiranje teksta

- podpora za Unicode nabore znakov
- vgrajeni pomnilniški pogoni (angl. storage engine) InnoDB, MyISAM, Merge, Memory, Federated, Archive, CSV, Blackhole, NDB Cluster

### Polno-tekstovno iskanje (angl. full-text search)

Večino iskanj izvajamo tako, da v WHERE stavku podamo pogoje, pri katerih naj nam iskanje izbere vrstico tako, da primerja vrednosti. Polno-tekstovno iskanje je popolnoma drugačno od običajnega iskanja po podatkovni bazi. Bolj je podobno iskanju, ki ga opravljajo spletni iskalniki. [41] Pri polno-tekstovnem iskanju generiramo posebno vrsto indeksa, kjer se namesto primerjanja vrednosti išče ključne besede v besedilu tako, da jih razvrščamo po relevanci. Polno-tekstovno iskanje deluje tudi brez ustreznih polno-tekstovnih indeksov, vendar pa konstrukcija takih indeksov bistveno pospeši delovanje polno-tekstovnih poizvedb. Primer sintakse polno-tekstovnega iskanja lahko vidimo na sliki 3.5. Iščemo lahko po enem ali več tekstovnih stolpcih znakovnih tipov, kot so CHAR, VARCHAR in TEXT.

```
public List<Mkb> booleanSearchLemeSlovenskiMkb(String iskalnaFraza){
  EntityManager emN = emf.createEntityManager();
  Query query = emN.createNativeQuery(
      "SELECT * FROM MKB WHERE MATCH(slovenski_naziv, concat_lemas) AGAINST(?1 IN BOOLEAN MODE)",
      Mkb.class
      );
  query.setParameter(1, iskalnaFraza);
  List<Mkb> listaMkbjev = query.getResultList();
  return listaMkbjev;
}
```

Slika 3.5: Uporaba ogrodja Hibernate za izvedbo SQL poizvedbe polno-tekstovnega iskanja in pozivanje rezultatov v javanske objekte.

Ločimo dve različici iskanja po polno-tekstovnem indeksu. **Polno-tekstovno iskanje v načinu naravnega jezika** (angl. natural language mode) in iskanje v načinu Boolove algebre. Pri poizvedbi polno-tekstovnega iskanja v naravnem jeziku določamo relevanco vsake vrstice za poizvedbo. Relevanco določimo na osnovi števila in frekventnosti besed, ki se ujemajo. Besede, ki

se v indeksu pojavljajo redkeje, bolj prispevajo k relevantnosti ujemanja, kot tiste, ki se pojavljajo pogosteje. Besede, ki se pojavljajo v več kot polovici vrstic, ne prispevajo k relevantnosti ujemanja. Pri iskanju v **načinu Boolove algebre** relevanco besede pri ujemanju določa sama poizvedba. Po potrebi lahko za besede določimo, da se morajo ali da se ne smejo pojavljati.

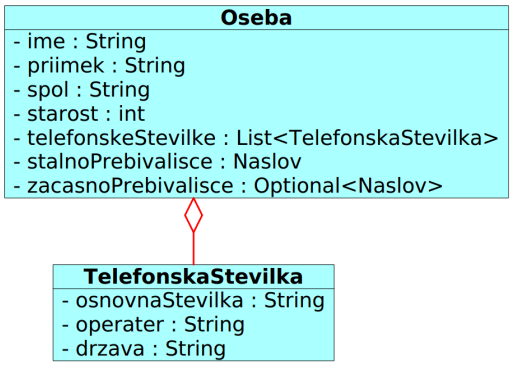
Ta način določanja zadetkov, pri dani iskalni frazi, je znan pod kratico tf-idf (angl. term frequency–inverse document frequency), [23] podrobnosti MySQL-ove implementacije pa so opisane v pripadajoči dokumentaciji. [17]

### 3.3.3 Objektno–relacijska preslikava

V računalništvu je objektno–relacijska preslikava (ORM) uveljavljena tehnika za pretvorbe med objektno naravnanimi programskimi jeziki in relacijskimi podatkovnimi bazami. Namen te tehnologije je poenostaviti dostop do podatkovne baze, skrajšati čas razvoja, olajšati vzdrževanje, izboljšati upravljanja z viri kot so povezave na bazo in uporaba predpomnilnika. Z uporabo ORM tehnologije dobimo virtualne objektne podatkovne baze. Pri objektno usmerjenemu programiranju so spremenljivke objekti, ki običajno nimajo svojega ekvivalenta v relacijskem svetu podatkovnih baz. ORM tehnologija nam omogoča preslikavo iz relacijskega sveta podatkovnih baz v objektni svet programskih jezikov. Obstaja tako prosto dostopna kot plačljiva ORM programska oprema.

Delovanje ORM programske opreme lahko ponazorimo z naslednjim primerom. Imamo objekt Oseba, ki ima različne atribute kot so npr. ime, priimek, spol, starost, seznam telefonskih številk, stalno prebivališče, začasno prebivališče, itd. Seznam telefonskih številk je predstavljen s poljem kazalcev, kjer vsak kazalec kaže na objekt TelefonskaStevilka (slika 3.6). Vsak objekt ima razen atributov tudi različne metode, kot so npr. metode za nastavljanje vrednosti atributov in metode, ki vračajo vrednost atributov. Mnoge od pogosto uporabljanih podatkovnih baz so upravljane s sistemi za upravljanje podatkovnih baz (SUPB), ki lahko hranijo samo preproste spremenljivke kot so cela števila in nizi, ki so organizirani v tabele. Tako metode in kazalci na

druge objekte, ki pripadajo nekemu objektu, nimajo svojega ekvivalenta v relacijskem svetu. Če programer želi shraniti objektno spremenljivko v podatkovno bazo, jo mora najprej pretvoriti v skupino preprostih spremenljivk in obratno. Če imamo npr. objekt TelefonskaStevilka in ga želimo shraniti v podatkovno bazo, ga ne moremo shraniti kot celoto ampak moramo shraniti vsako od njegovih polj posebej npr. osnovnaStevilka, operater in drzava. Ta proces avtomatizirajo sistemi za objektno–relacijsko preslikavo [29].




Slika 3.6: Razred Oseba agregira razred TelefonskaStevilka. Metode za nastavljanje in vracanje vrednosti so izpuščene.

### 3.3.4 Načelne razlike v objektnem in relacijskem modelu

Ključne načelne razlike med objektno orientiranim programiranjem in relacijskimi podatkovnimi bazami so naslednje: [20]

- **Relacijski in objektni svet** — v relacijskem svetu imamo opravka s podatki organiziranimi v relacije, v svetu objektnega programiranja pa imamo opravka z metodami, ki sledijo neki logiki in podatke obdelajo. Koncepti iz sveta objektno usmerjenega programiranja, kot so dedovanje in polimorfizem, v relacijskem svetu niso podprti.

- **Relacijska shema** — v relacijskem svetu obstajajo odvisnosti, ki sledijo iz relacijske sheme in nimajo svojega ekvivalenta v objektnem svetu. V relacijskem svetu ima lahko določena entiteta neke entitetne tabele obvezno povezavo z neko drugo entiteto neke točno določene druge entitetne tabele. Še najbližji približek takih povezav v objektnem svetu je dedovanje, ki pa je strukturirano kot drevo in ni obvezno, saj lahko vsak objekt nadomestimo s praznim kazalcem.

- **Dostopna pravila** — v relacijskih podatkovnih bazah je način dostopa do podatkov vnaprej določen z relacijsko shemo. Striktne razlike med javnimi in zasebnimi podatki nimamo. Podatkovno domeno lahko v SUPB-ju določimo bolj natančno, v OOP svetu pa definiramo poljubne metode, ki lahko podatke skoraj poljubno spremenijo.

- **Istovetnost objektov** — za objekte smatramo, da imajo enolično identiteto. Dveh objektov, ki imata slučajno vse atribute enake, ne smatramo za identična. Objekt je enolično določen z lokacijo v računalnikovem spominu, ki je določena s kazalcem. Pri relacijah je enoličnost definirana drugače. Dve različni entiteti ne moreta imeti vseh atributov enakih.

- **Razlike v podatkovnih tipih** — en tak primer razlik v podatkovnih tipih so nizi. V podatkovnih bazah so nizi navadno fiksne dolžine, kar je ugodno za zmogljivost podatkovne baze. V objektno usmerjenem programiranju pa je spremenljivka, s katero referenciramo niz, samo kazalec na prostor v spominu, kjer se niz nahaja.

- **Normalne oblike podatkovnih baz** — normalizacije relacijskih podatkovnih baz v objektnem programiranju pogosto ne upoštevamo. V normaliziranih relacijskih podatkovnih bazah npr. ne dovolimo parcialnih odvisnosti in relacijo s parcialno odvisnostjo razbijemo na dve relaciji, po drugi strani pa tako odvisnost v objektnem programiranju lahko dovolimo.

### Težave pri objektno–relacijski preslikavi

Razlike med objektnim in relacijskim modelom lahko premostimo z objektno relacijsko preslikavo. Ali to pomeni, da poznavanje strukturiranega povpraševalnega jezika (SQL) dandanes ni več pomembno?

Odgovor je seveda—ne. Gotovo drži, da osnovne funkcionalnosti SQL-a lahko opravlja ORM. Kljub temu pa so za visokokvalitetno aplikacijo, v nasprotju z aplikacijo, ki samo deluje, ključni določeni napredni SQL koncepti. Sledi nepopoln seznam primerov uporabe pri katerih se ORM-ji v splošnem še vedno ne obnesejo najbolje ali pa navedene funkcionalnosti sploh ne podpirajo: [10]

- rekurzivne ali hierahične poizvedbe
- uporabniško določeni podatkovni tipi
- tipi specifični za neko računalniško okolje
- paketno pisanje/posodabljanje/brisanje podatkovne baze
- upravljanje z napakami
- rezultati zunanjih stikov se slabo prilegajo objektnemu modelu
- modularizacija shranjenih procedur
- *učinkovite* ostranjene poizvedbe (poizvedbe, pri katerih določimo zaporedno številko prvega vrnjenega elementa in število vseh vrnjenih elementov)
- poizvedbe, ki temeljijo na funkcijah kot so rownum, rank in parititons

Tudi za učinkovito uporabo ORM orodij moramo izhajati iz dobrega razumevanja relacijskega modela in SQL-a. Poznati moramo relacijski model in imeti znanja kot je npr. normalizacija, da lahko zagotovimo neokrnjenost podatkov, prav tako je dobro poznavanje SQL-a potrebno za izboljšanje zmogljivosti ORM-ja. ORM avtomatizira mnogo ponavljajočega kodiranja,

kljub temu pa pri učinkoviti uporabi modernih SUPB-jev brez poznavanja relacijske tehnologije ne gre. [32]

**Ali se moramo odločiti za specifično ORM implementacijo in implementacijo podatkovne baze?**

Na prvi pogled se zdi, da se nam ni potrebno odločiti za specifično implementacijo ORM-ja. Programski vmesniki, kot je javanski trajnostni programski vmesnik (JPA), navidezno omogočajo preprosto menjavo implementacije ORM-ja. JPA je na nek način podoben standardu za povezljivosti podatkovnih baz (ODBC). JPA in ODBC ponujata skupen programski vmesnik (API) do različnih podatkovnih baz. Slabost je seveda, da standardni API ne podpira funkcionalnosti, ki so specifične za posamezne ORM-je v primeru JPA-ja in SUPB-je v primeru ODBC-ja.

Z minimalnimi prilagoditvami kode lahko zamenjamo tudi podatkovno bazo, ki jo uporabljamo. Podatkovna baza je z ORM-jem povezana preko gonilnika, [16] ki ima implementiran aplikacijski programski vmesnik JDBC (angl. java database connectivity). JDBC je objektno usmerjena implementacija povezovanja s podatkovno bazo v duhu ODBC. [36]

Glavna prednost uporabe skupnega programskega vmesnika je, da lahko načeloma neko implementacijo ORM-ja kasneje zamenjamo z neko drugo implementacijo. Podobno lahko podatkovno bazo zamenjamo z neko drugo podatkovno bazo. Ta prednost pa se povsem izgubi, če upoštevamo, da v praksi v večini primerov implementacije podatkovne baze in ORM-ja ne zamenjamo. Dostikrat se zgodi, da namesto da bi en ORM zamenjali z drugim, razvijalci v primeru ozkega grla v zmogljivosti ORM v celoti ali delno nadomestijo z direktnimi SQL klici. Torej nam ne preostane drugega, kot da se poglobimo v napredne funkcionalnosti SQL in funkcionalnosti, ki jih nek ORM ponuja. V našem primeru je to Hibernate. [9]

### 3.3.5 Hibernate - objektno–relacijsko preslikovalno ogrodje

Uporaba Hibernata nam ponuja naslednje prednosti [32]

- *Produktivnost*—z uporabo Hibernata se izognemo pisanju veliko ponavljajoče se kode. Na ta način lahko pomembno zmanjšamo čas, ki je potreben za razvoj aplikacije.

- *Olajšano vzdrževanje*—Avtomatizirana objektno relacijska preslikava zmanjša število vrstic kode potrebne za razvoj aplikacije. Tak sistem je *bolj razumljiv* in *bolj preprost za prestrukturiranje*.

- *Zmogljivost*—Podobno kot lahko koda napisana v zbirnem jeziku deluje hitreje kot koda napisana v Javi, je možno ročno napisati kodo, ki zagotavlja trajnost in je hitrejša od optimizacij, ki jih avtomatsko uporablja Hibernate. Kljub temu pa je to v praksi zelo težko doseči, saj Hibernate uporablja veliko optimizacij *naenkrat*. En tak primer je predpomnjenje, ki je v Hibernatu učinkovito in ga je možno tudi enostavno dodatno nastaviti. Tako lahko razvijalci svoj čas uporabijo za to, da ročno optimizirajo tistih nekaj ozkih grl, ki še ostanejo.

- *Neodvisnost od ponudnika*— Hibernate lahko zmanjša tveganja povezana s preveliko odvisnostjo od nekega ponudnika. Tudi v primeru, ko ne planiramo menjave SUPB-ja nam ORM omogoča določeno *stopnjo prenosljivosti*. Podobno nam ORM pride prav, če v razvojnem okolju uporabljamo drug SUPB kot v produkcijskem.

### Sestavni deli orodja Hibernate

Hibernate je obširen projekt, katerega cilj je obvladati problem upravljanja trajnih podatkov v Javi. Hibernate pravzaprav ni samo ORM, ampak je zbirka orodij za upravljanje s podatki, katerih funkcionalnost daleč presega ORM. Primer uporabe ogrodja Hibernate za dostop do podatkovne baze lahko vidimo na sliki 3.7.

Projekt Hibernate sestavljajo naslednji deli: [32]

- *Hibernate ORM*—Hibernate ORM predstavlja jedro tega projekta in vsebuje osnovne storitve, ki omogočajo trajnost podatkov v relacijskih

```
public List<Mkb> ostranjenaTabelaMkb(int odmik, int max)
{
  EntityManager emN = emf.createEntityManager();
  List<Mkb> stran = emN.createQuery("from Mkb m", Mkb.class).setFirstResult(odmik).setMaxResults(max).getResultList();
  emN.close();
  return stran;
}
```

Slika 3.7: Uporaba ogrodja Hibernate za ostranjevanje MySQL tabele MKB-jev.

podatkovnih bazah. Hibernate ORM ima tudi lastni in lastniški programski vmesnik. Hibernate ORM je možno uporabljati neodvisno od drugih delov orodja Hibernate. Preko lastnega API-ja ga lahko uporabljamo z vsakim Java EE/J2EE aplikacijskim strežnikom in v Swing aplikacijah, v preprostem servletu itd. Vse kar potrebujemo za uporabo Hibernate ORM-ja je, da lahko skonfiguriramo gonilnik podatkovnega vira.

- *Hibernate entitetni upravljalec*—To je Hibernatova implementacija javanskega programskega vmesnika za trajnost (JPA). Lahko jo postavimo nad Hibernate ORM. Še vedno pa imamo možnost direktno dostopati do Hibernate ORM-ja. Hibernatova lastna funkcionalnost je nadmnožica funkcionalnosti, do katerih lahko dostopamo z JPA-jem.

- *Hibernate preverjalnik*—To je referenčna implementacija preverjanja javanskega zrna (specifikacija JSR 303), ki nam omogoča, da z anotacijami definiramo model metapodatkov.

- *Hibernate Envers*—Envers je namenjen revizijskemu beleženju (angl. audit logging) in ohranjanju več verzij podatkov v podatkovni bazi SQL. Tako lahko beležimo zgodovino podatkov in omogočimo aplikaciji, da za sabo pusti sled, ki jo lahko ob morebitni reviziji uporabimo.

- *Hibernate Search*—Hibernate Search indeksira podatke o domenskem modelu in podatke hrani v podatkovni bazi Apache Lucene. Omogoča nam preiskovanje te podatkovne baze z zmogljivim in lastnim program-

skim vmesnikom. Mnogi projekti uporabljajo Hibernate Search, da Hibernate ORM-ju omogočijo iskanje po tekstu. Glede na to, da MySQL že privzeto podpira polno-tekstovno iskanje, Hibernate Searcha in Apache Lucena nismo uporabili.

- *Hibernate OGM*—Hibernate OGM je najnovejši dodatek orodju Hibernate in omogoča JPA podporo za razširjeni strukturirani povpraševalni jezik za delo s podatkovnimi bazami (NoSQL). Hibernate OGM nam omogoča shranjevanje entitet v obliki ključ/vrednost, v dokumentni obliki ali v obliki grafov.

## 3.4 Računalniška obdelava naravnega jezika

### 3.4.1 Osnovni pojmi

Računalniško obdelavo besedila v naravnem jeziku pogosto začnemo z razčlenjevanjem (angl. tokenization). [8] Razčlenjevanje je postopek, pri katerem zaporedje znakov, ki tvori besedilo, razdelimo na besede, ločila, številke in druge elemente. Na tej stopnji besede niso razvrščene v slovnične kategorije. Kljub temu pa lahko marsikaj izvemo iz razmeroma preproste analize razčlenjenega besedila.

Razčlenjevanju lahko sledi krnjenje (angl. stemming) ali lematizacija. [26] Krnjenje je postopek, ki besedi odvzame končnico, beseda ,,hoditi“ ima krn ,,hodi“. Lematizacija je postopek, pri katerem besedi določimo njeno osnovno (slovarsko) obliko, ki jo imenujemo lema. V mnogih jezikih, tudi v slovenščini, se besede pojavljajo v različnih skladenjskih oblikah, lematizacija pa besedam pripiše osnovno obliko. Besedam ,,hodim“, ,,hodiš“, ,,hodimo“, vsem pripada lema ,,hoditi“. Razmerje med lematizacijo in krnjenjem ni v vseh jezikih enako. V angleškem jeziku sta lematizacija in krnjenje približno enako uporabna, v slovenskem jeziku pa so krni pogosto prekratki in se zlijejo s krni drugih besed.

### 3.4.2 Zgodovina

Računalniška obdelava naravnega jezika ima dolgo zgodovino, ki se začenja v petdesetih letih prejšnjega stoletja. [28]

Prva demonstracija strojnega prevajanja je bila rezultat sodelovanja med univerzo Georgetown in podjetjem IBM. [25] V okviru te demonstracije je bil 7. januarja 1954 prikazan strojni prevod šestdesetih ruskih stavkov v angleščino. Eksperiment je bil zasnovan z namenom pritegniti pozornost javnosti in posledično pridobiti finančna sredstva. Po trditvah avtorjev naj bi bil strojni prevod realnost v samo treh do petih letih. Šestdeset let kasneje ta problem še vedno ni zadovoljivo rešen. Po desetih letih so bili raziskovalci še vedno daleč od cilja, financiranje tega področja pa se je bistveno zmanjšalo.

Področje je svoj preporod doživelo šele konec osemdesetih let prejšnjega stoletja, ko so bili razviti prvi sistemi za statistično strojno prevajanje. Osemdeseta leta prejšnjega stoletja so tudi prelomnica za uporabo tehnik strojnega učenja za obdelavo naravnega jezika. Prvi primeri uporabe strojnega učenja za obdelovanje naravnega jezika so bila odločitvena drevesa, ki so bila zapisana v obliki if-stavkov.

#### Uporaba strojnega učenja za obdelavo naravnega jezika

Sodobni NLP algoritmi temeljijo na strojnem učenju, najpogosteje na statističnem strojnem učenju. [28] Statistično sklepanje se uporablja za strojno učenje zakonitosti jezika in temelji na analizi velikega števila resničnih primerov. Za namene računalniške obdelave naravnega jezika je bilo uporabljenih mnogo različnih algoritmov s področja strojnega učenja. Vhodne podatke ponavadi predstavlja ogromna kopica značilk, ki so generirane iz vhodnih podatkov.

Prvi algoritmi so uporabljali odločitvena drevesa, ki so bila togi del programa. Sčasoma pa so se raziskave osredotočile na statistične modele, ki uporabljajo prilagodljive, verjetnostne odločitve, ki temeljijo na pripisovanju uteži različnim značilkam.

Algoritmi strojnega učenja imajo naslednje prednosti pred togo zapisanimi algoritmi: [28]

- postopki, ki jim sledimo pri algoritmih strojnega učenja nam omogočajo, da se samodejno osredotočimo na najbolj običajne primere, kadar pa smo pravila odločanja napisali ročno, sledimo občutku, ki pa ni nujno najbolj primeren za karakterizacijo najbolj običajnih primerov.

- pri strojnem učenju lahko z uporabo statističnega sklepanja pridemo do modelov, ki so odporni na neznane vhodne podatke – to je na besede ali besedne zveze, na katere še nismo naleteli. V splošnem je za take primere izjemno težko ročno napisati pravila, hkrati pa je podvrženo napakam in časovno zahtevno.

- natančnost sistemov, ki temeljijo na strojnem učenju, lahko izboljšamo tako, da izberemo večjo učno množico. Pri ročno napisanih pravilih lahko dosežemo večjo točnost samo tako, da napišemo bolj kompleksna pravila, kar je bistveno težje od tega, da vzamemo večjo učno množico.

### 3.4.3 Področja uporabe NLP

Področja, na katerih se računalniška obdelava naravnega jezika največ uporablja, so naslednja [28]:

- **Samodejno povzemanje** Cilj samodejnega povzemanja je narediti berljiv povzetek nekega teksta. Pogosto je tip besedila, ki ga povzemamo, znan vnaprej, kot npr. finance, zdravje, politika ipd.

- **Razločevanje referenc (angl. coreference resolution)** Razločevanje referenc je določanje besed, ki se nanašajo na enake objekte.

- **Diskurzivna analiza** Pod pojmom diskurzivna analiza razumemo več sorodnih nalog. Ena od nalog je določitev strukture diskurza teksta – to je določitev diskurzivnega razmerja med stavki kot je razlaga, podajanje podrobnosti, podajanje nasprotja. Naslednja naloga diskurzivne

analize je prepoznavanje govora v tekstu, npr. vprašanje tipa da ali ne, izjava ipd.

- **Strojno prevajanje** Samodejno prevajanje teksta iz enega naravnega jezkia v drugega je eden najtežjih problemov umetne inteligence, saj za popolno rešitev tega problema potrebujemo znanje s področja slovnice, semantike, dejstev iz resničnega sveta itd.

- **Oblikovna segmentacija** Oblikovna segmentacija razdeli besede v posamezne morfeme in v različne kategorije. Težavnost oblikovne segmentacije je močno odvisna od kompleksnosti jezika, katerega besede segmentiramo. Angleščina je morfološko dokaj preprosta, kar pa ne velja za slovenščino, ki ima šest sklonov in tri slovnična števila.

- **Prepoznavanje lastnih imen** Pri prepoznavanju lastnih imen gre za to, da v danem tekstu določimo besede, ki ustrezajo določenim objektom kot so osebe, kraji, organizacije ipd. Kljub temu, da nam pomaga pisanje z veliko začetnico, ta informacija sama po sebi ni dovolj, da bi enolično določili povezavo med neko besedo in objektom – npr. prvo besedo v stavku v vsakem primeru pišemo z veliko začetnico.

- **Generiranje naravnega jezika** Pri generiranju naravnega jezika gre za pretvorbo informacij, ki so shranjene v podatkovnih bazah, v berljiv naravni jezik.

- **Razumevanje naravnega jezika** Pri razumevanju naravnega jezika gre za pretvorbo teksta v bolj formalno zapisane logične strukture, s katerimi računalniki lažje upravljajo. Razumevanje naravnega jezika pomeni ugotavljanje zamišljenega semantičnega pomena iz več možnih semantičnih pomenov, ki jih je možno pripisati določeni kombinaciji besed.

- **Optično prepoznavanje znakov** Pri optičnem prepoznavanju znakov moramo za dano sliko ugotoviti tekst, ki ga ta slika predstavlja.

- **Oblikoslovno označevanje (angl. part-of-speech tagging)** Za dani stavek moramo vsaki besedi določiti njeno oblikoslovno kategorijo. Slovenščina je zaradi slovničnih sklonov manj dvoumna kot npr. angleščina, kjer je sklanjanje manj poudarjeno.

- **Razčlenjevanje (angl. parsing)** Pri razčlenjevanju gre za določitev razčlenitvenega drevesa danega stavka. Slovnica naravnega jezika je lahko dvoumna, saj ima lahko nek stavek več možnih interpretacij. Za tipični stavek je možno sestaviti tisoče razčlenitvenih dreves, večina teh pa je človeku popolnoma nesmiselnih.

- **Odgovor na vprašanje** Pri generaciji odgovora na vprašanje za dano vprašanje v naravnem jeziku najdemo odgovor. Določena vprašanja imajo točno določene pravilne odgovore, kot npr. katero je glavno mesto določene države, nekatera vprašanja pa so bolj odprte narave, kot je naprimer vprašanje kaj je smisel življenja (to je seveda samo v primeru, če za odgovor ne sprejmemo, da je smisel življenja število 42).

- **Izluščenje povezave** Pri izluščenju povezave za določen tekst ugotovimo povezave med imenovanimi objekti npr. v kakem sorodu sta dve osebi.

- **Razločevanje mej stavka** Pri razločevanju mej stavka poskušamo za dani tekst ugotoviti, kje se določen stavek konča in drug stavek začne. Sicer so stavki običajno ločeni z ločili, vendar lahko ta ločila uporabljamo tudi v druge namene, npr. za okrajšave.

- **Analiza tonalnosti besedila (angl. sentiment analysis)** Pri analizi tonalnosti besedila poskušamo izluščiti informacije v povezavi z odnosom avtorja do vsebine besedila.

- **Razpoznavanje govora** Pri razpoznavanju govora je potrebno iz zvočnega zapisa govora ene ali več oseb zapisati besedilo. Ta problem je ravno

nasproten problemu generiranja naravnega jezika. Podproblem razpoznavanja govora je razčlenjevanje govora (angl. speech segmentation), kjer gre za izluščevanje besed iz audio zapisa.

- **Razpoznavanje teme** Pri razpoznavanju teme gre za določitev segmentov, ki so posvečeni različnim temam in določitev teme vsakega segmenta.

- **Določanje pomena besede** Pri določanju pomena besede gre za določitev pomena besede, ki je enakopisnica neke druge besede.

### 3.4.4 Lematiziranje z uporabo orodja LemmaGen

Lema je kanonska oziroma citatna oblika besede kot npr. gesla v slovarju. Lematizacija je postopek, po katerem spremenimo besedo v njeno kanonično obliko. Večpomenskost besed lahko pri lematizaciji predstavlja težave. Tako je lahko npr. lema besede ,,hotela“ lahko samostalnik ,,hotel“ ali pa glagol ,,hoteti“. Dodatne težave pri lematizaciji predstavljajo tudi morfološko bogati jeziki z veliko slovničnimi oblikami, kot je slovenski jezik, kjer se lahko krn poveže z veliko različnimi končnicami. Končnice in krni so v razmerju mnogo proti mnogo, izbira ustrezne končnice pri danem krnu pa je odvisna od mnogo dejavnikov, od fonoloških do semantičnih. Torej za določitev ustrezne leme neki besedi potrebujemo kontekst, v katerem se ta lema nahaja. Lematizacija je koristna pri mnogih procesih kot je npr. podatkovno rudarjenje ali pa razčlenjevanje iskalne fraze. LemmaGen [38] je eden od obstoječih lematizatorjev, ki je implementiran za različne naravne jezike in v različnih programskih jezikih, med drugim tudi v Javi [14].

# Poglavje 4

# Rezultati

Končni izdelek diplomskega dela je aplikacija Iskanje mednarodne klasifikacije bolezni (IMKB), ki omogoča iskanje po mednarodnih klasifikacijah bolezni. V okviru tega dela je nastala programska koda v programskih jezikih JavaScript, SQL in Java, pri čemer je javanska koda organizirana po principu meja, nadzor in entiteta (angl. boundary, control, entity), ki ga je populariziral Ivar Jacobson. [37]

Aplikacija IMKB, ki je nastala v okviru diplomskega dela, uporabniku pomaga določiti ustrezno mednarodno klasifikacijo bolezni, kar je mednarodni standard za izmenjavo podatkov o boleznih in zdravstvenih stanjih. Podatke smo pridobili iz tabele v formatu Microsoft Excel. [3] S pomočjo knjižnice Apache POI pa smo jih prenesli v podatkovno bazo, ki se nahaja na sistemu za upravljanje s podatkovnimi bazami MySQL. Nad temi podatki uporabniku ponujamo več različnih iskanj. Uporabnik aplikacijo uporablja tako, da v vnosno polje vtipka celotno ali delno diagnozo, aplikacija pa izpiše seznam MKB-jev, ki ustrezajo povpraševanju. Kljub temu, da je iskani MKB običajno izpisan na vrhu, pa v primeru, ko temu ni tako, uporabniku priskoči na pomoč filtrirno polje, ki dodatno filtrira rezultate po kategoriji, podkategorijah in slovenskem opisu bolezenskega stanja.

Največji izziv diplomske naloge je bil zagotoviti dovolj hitro iskanje ustreznih MKB-jev na nivoju relacijske podatkovne baze. Najprej smo preizkusili

iskanje pri katerem smo imeli leme MKB-jev zapisane v svoji tabeli. Leme in MKB-ji so v razmerju mnogo proti mnogo, zato je med tema tabelam še povezovalna tabela. Ustrezne MKB-je za neko lemo dobimo tako, da najprej naredimo stik med tabelo lem in povezovalno tabelo, potem pa še med povezovalno tabelo in tabelo MKB-jev. Javanska metoda, ki smo jo klicali za izvajanje te operacije, je potrebovala približno desetinko sekunde.

Kot alternativni način iskanja smo preizkusili polno-tekstovno iskanje po stolpcu z opisi MKB-jev. Tako iskanje je trajalo manj kot desetinko sekunde (tabela 5.1). Slabost tega iskanja je, da so rezultati manj povezani z iskalno frazo, kot pri prvem iskanju.

Prednosti obeh iskanj smo združili tako, da smo tabeli, v kateri so shranjeni MKB-ji, pridružili stolpec, v katerega smo naložili vse leme, ki jih nek MKB vsebuje. Na pridruženem stolpcu in v stolpcu z opisi MKB-jev smo vršili polno-tekstovno iskanje. Na ta način smo dobili rezultate v času, ki je potreben za polno-tekstovno iskanje (manj kot desetinka sekunde), hkrati pa smo dosegli kvaliteto razvrščanja MKB-jev, ki je primerljiva z iskanjem po tabeli lem.

Aplikacijo (slika 4.1) uporabljamo tako, da v iskalno polje vpišemo frazo, ki je sestavljena iz besed, s katerimi želimo določiti iskano klasifikacijo. Pod iskalnim poljem je stikalo s katerim izberemo način iskanja, ki želimo, da je uporabljen nad klasifikacijami. Na izbiro so naslednji trije načini razvrščanj klasifikacij:

- **Iskanje 1 – po lemah:** gre za iskanje po opisu MKB-jev. Vsak opis MKB je razdeljen na besede, ki so lematizirane s pomočjo orodja LemmaGen in spravljene v MySQL tabeli. Pri tem tipu iskanja uporabnik vpiše v iskalno okence frazo, ki je del diagnoze. Aplikacija pošlje vpisano frazo na strežnik, kjer se zahteva razdeli na besede, vsaka beseda pa se lematizira. Vsako od lem poiščemo v prej pripravljeni MySQL tabeli in preko dvojnega stika določimo ustrezne MKB-je. Vsem MKB-jem določimo točkovanje ustreznosti. Za vsako lemo, ki je bila v vhodnem nizu in hkrati pripada nekemu MKB-ju se MKB-ju pripiše to-


Slika 4.1: Iskanje mednarodne klasifikacije bolezni za diagnozo ,,tuberkoloza na pljučih“.

liko točk, kolikor znaša število črk leme na peto potenco. Ta način smo izbrali popolnoma arbitrarno in z namenom, da dolgim lemam pripišemo večji pomen kot več kratkim lemam. MKB-je razvrstimo po številu točk. Od strežnika do odjemalca potuje 100 MKB-jev z najvišjim številom točk.

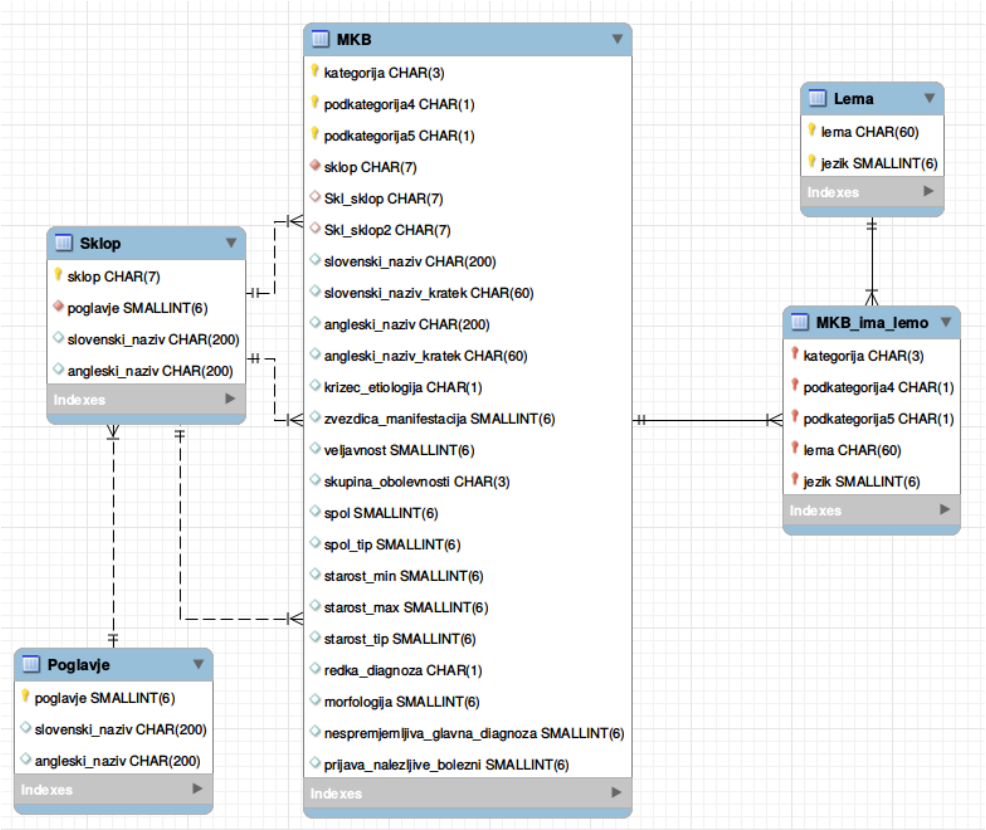
- **Iskanje 2 – polno-tekstovno iskanje v naravnem jeziku po opisih:** uporabili smo funkcionalnost, ki jo MySQL ponuja pod imenom ,,natural language full-text search“, kjer iščemo ujemanje iskanega niza s podatki, ki so shranjeni v podatkovni bazi. Primerjavo delamo po enem ali več stolpcih. Iskanje po enem ali več stolpcih je potrebno predhodno omogočiti z izgradnjo indeksa besed. Privzeto je iskanje, ki je neodvisno od tega ali so črke velike ali male tiskane. Iskanje avtomatično vrne vrstice razporejene po relevantnosti, najprej vrne najbolj relevantne vrstice.

- **Iskanje 3 – polno-tekstovno iskanje po opisih in lemah:** to iskanje je podobno iskanju v naravnem jeziku s to razliko, da smo uporabili funkcionalnost, ki jo MySQL ponuja pod imenom ,,boolean full-text search“. Tu smo za posamezne besede, ki sestavljajo iskalni niz, določili ali jih MKB mora ali pa jih ne sme vsebovati. Za namen tega iskanja smo dodali stolpec v tabelo MKB-jev in vanj vpisali vse leme, ki določenemu MKB-ju pripadajo. Iskalni niz smo razbili na besede in jih lematizirali. Te leme smo uporabili za iskanje po stolpcu, v katerem imamo leme, ki določenemu MKB-ju pripadajo, ter po stolpcu slovenski_naziv.

Ko nam eno od zgoraj opisanih razvrščanj vrne MKB-je, se le-ti izpišejo v brskalnik. Za vsak MKB je izpisana njegova kategorija, dve podkategoriji in njegov opis v slovenskem jeziku. S tipkanjem v polje filter se prikaz dinamično prilagaja vpisani frazi tako, da ostanejo prikazani samo tisti MKB-ji, ki vsebujejo vtipkani niz. S klikom na katero od vrstic z MKB-ji se nam na desni strani prikažejo podrobnosti za dotični MKB.

## 4.1 Pridobivanje podatkov

Slovensko verzijo mednarodne klasifikacije bolezni smo pridobili v obliki preglednic v formatu Microsoft Excel. [3] Podatki o mednarodnih klasifikacijah bolezni so organizirani v poglavja, vsako poglavje ima več sklopov. Vsak sklop pa ima mnogo vpisov posameznih opisov bolezenskih stanj in drugih okoliščin v katerih se pacienti znajdejo v stiku z zdravstvenimi delavci. Shematska predstavitev zapisa podatkov v podatkovni bazi je podana na sliki 4.2.




Slika 4.2: Shema podatkovne baze

### 4.1.1 Integracija podatkov v aplikacijo IMKB

Iz Excelovih preglednic smo podatke prebrali z javanskim API-jem za branje dokumentov podjetja Microsoft – Apache POI (slika 4.3). Iz preglednic smo prebrali ustrezne podatke za tabele Poglavje, Sklop in MKB (slika 4.4).

```
private HSSFWorkbook odpriExcelTabelo(String path){
  HSSFWorkbook wb = null;
  try{
    InputStream inp = new FileInputStream(path);
    wb = new HSSFWorkbook(new POIFSFileSystem(inp));

  }catch(FileNotFoundException e){
    System.out.println(e.toString() + "Main.java metoda odpriExcelTabelo FileNotFoundExeption");
  }catch(IOException e){
    System.out.println(e.toString() + "Main.java metoda odpriExcelTabelo IOException");
  }
  return wb;
}
```

Slika 4.3: Uporaba Microsoft Excel tabele s pomočjo ogrodja Apache POI.

Tabelo Leme smo pridobili tako, da smo za posamezni MKB njegov slovenski naziv razbili na posamezne besede, jih z uporabo orodja LemmaGen lematizirali, vpisali v tabelo lem, hkrati pa smo gradili še povezovalno tabelo med tabelo Leme in tabelo MKB. Podatke smo iz aplikacije do podatkovne baze prenašali s pomočjo ogrodja Hibernate ORM. Med odjemalcem in aplikacijo pa smo podatke prenašali s pomočjo ogrodja Jackson, [11] ki je poskrbelo za pripravo podatkov za transport (angl. marshalling) v JSON notaciji in s pomočjo ogrodja RESTEasy, [22] ki je poskrbelo za pošiljanje in prevzemanje podatkov preko HTTP metod, v našem primeru smo RESTEasy uporabljali za POST in GET zahtevke. Primer kode, kjer si pomagamo z ogrodji RESTEasy in Jackson, je na sliki 4.5.

```
public void napolniTabeloMkb(){
   //odpri excell tabelo s kodami
   HSSFWorkbook wb      = odpriExcelTabelo(path);
   HSSFSheet sheetKode = wb.getSheet("kode");
   int steviloVrsticKod    = sheetKode.getPhysicalNumberOfRows();
   // i zacnem stet z ena, ker je prva vrstica header poglavja
   for(int i=1;i<steviloVrsticKod; i++){
      HSSFRow vrsticaKoda = sheetKode.getRow(i);
      String kategorija                 = vrsticaKoda.getCell(1).toString();
      char podkategorija4               = vrsticaKoda.getCell(2).toString()=="" ? "x".charAt(0):vrsticaKoda.getCell(2).toString().charAt(0);
      char podkategorija5               = vrsticaKoda.getCell(3).toString()=="" ? "x".charAt(0):vrsticaKoda.getCell(3).toString().charAt(0);
      String sklop1id                   = vrsticaKoda.getCell(4).toString();
      String sklop2id                   = vrsticaKoda.getCell(5).toString();
      String sklop3id                   = vrsticaKoda.getCell(6).toString();
      String slovenskiNaziv             = vrsticaKoda.getCell(9).toString();
      String slovenskiNazivKratek       = vrsticaKoda.getCell(10).toString();
      String angleskiNaziv              = vrsticaKoda.getCell(11).toString();
      String angleskiNazivKratek        = vrsticaKoda.getCell(12).toString();
      Short krizecEtiologija            = Short.parseShort((vrsticaKoda.getCell(13).toString().substring(0,1)));
      Short zvezdicaManifestacija       = Short.parseShort(vrsticaKoda.getCell(14).toString().substring(0,1));
      Short veljavnost                  = Short.parseShort(vrsticaKoda.getCell(15).toString().substring(0,1));
      String skupinaObolevnosti         = vrsticaKoda.getCell(16).toString();
      Short spol                        = vrsticaKoda.getCell(17).toString()=="" ? null : Short.parseShort(vrsticaKoda.getCell(17).toString().substring(0,1));
      Short spolTip                     = vrsticaKoda.getCell(18).toString()=="" ? null : Short.parseShort(vrsticaKoda.getCell(18).toString().substring(0,1));
      Short starostMin                  =  vrsticaKoda.getCell(19).toString()=="" ? null : Short.parseShort(vrsticaKoda.getCell(19).toString().substring(0,1));
      Short starostMax                  =  vrsticaKoda.getCell(20).toString()=="" ? null : Short.parseShort(vrsticaKoda.getCell(20).toString().substring(0,1));
      Short starostTip                  =  vrsticaKoda.getCell(21).toString()=="" ? null : Short.parseShort(vrsticaKoda.getCell(21).toString().substring(0,1));
      Short redkaDiagnoza               = vrsticaKoda.getCell(22).toString()=="" ? null : Short.parseShort(vrsticaKoda.getCell(22).toString().substring(0,1));
      Short morfologija                 =  vrsticaKoda.getCell(23).toString()=="" ? null : Short.parseShort(vrsticaKoda.getCell(23).toString().substring(0,1));
      Short nesprejemljivaGlavnaDiagnoza =  vrsticaKoda.getCell(24).toString()=="" ? null : Short.parseShort(vrsticaKoda.getCell(24).toString().substring(0,1));
      Short prijavaNalezljiveBolezni    =  vrsticaKoda.getCell(25).toString()=="" ? null : Short.parseShort(vrsticaKoda.getCell(25).toString().substring(0,1));

      Sklop sklop1 = interakcijaZBazo.najdiSklopPoIdju(sklop1id);
      Sklop sklop2 = interakcijaZBazo.najdiSklopPoIdju(sklop2id);
      Sklop sklop3 = interakcijaZBazo.najdiSklopPoIdju(sklop3id);
      MkbId mkbId = new MkbId(kategorija, podkategorija4, podkategorija5);
      Mkb mkb = new Mkb(mkbId, sklop2, sklop1, sklop3, slovenskiNaziv, slovenskiNazivKratek, angleskiNaziv, angleskiNazivKratek,
            krizecEtiologija, zvezdicaManifestacija, veljavnost, skupinaObolevnosti, spol, spolTip, starostMin, starostMax, starostTip,
            redkaDiagnoza, morfologija, nesprejemljivaGlavnaDiagnoza, prijavaNalezljiveBolezni, null);
      interakcijaZBazo.persistirajMKB(mkb);
   }

}
```

Slika 4.4: Uporaba ogrodja Apache POI za generacijo objektne predstavitve MKB-jev.

```
@Path("/storitve")
public class RestAuth {

   @POST
   @Path("/fullTextSearchMkb")
   @Produces(MediaType.APPLICATION_JSON)
   @Consumes(MediaType.APPLICATION_JSON)
   public ListMkbWrapper iskanjeFullTextPoMkbjih (String incoming)
   {
      long startTime = System.nanoTime();
      List<Mkb> mkbList = izb.fullTextSearchSlovenskiMkb(incoming);
      //vzemi samo 100 najboljsih
      if(mkbList.size() > 100){
         mkbList = mkbList.subList(0, 100);
      }
      long endTime = System.nanoTime();
      long duration = (endTime - startTime)/1000000;
      System.out.println("Iskanje po slovenskih nazivih IN NATURAL  LANGUAGE MODE je trajalo: " + duration);
      System.out.println("Iskalna fraza: " + incoming );

      return new ListMkbWrapper(mkbList);
   }
```

Slika 4.5: Uporaba ogrodij RESTEasy in Jackson za obdelavo POST zahtevkov.

# Poglavje 5

# Eksperimenti

Aplikacija, ki je rezultat tega diplomskega dela, je bila za testne namene nameščena na delovno postajo z naslednjo strojno opremo:

- **delovni pomnilnik:** 8GB, 1600MHz, DDR3
- **procesor:** Intel Core i5, 3.40GHz, štiri jedra

Aplikacija je bila nameščena na aplikacijski strežnik WildFly 8.2.1.Final (strežnik je odprtokoden, večino kode prispeva podjetje RedHat), ki je tekel na operacijskem sistemu Linux (distribucija Ubuntu 16.04 LTS, podjetje Canonical Ltd).

## 5.1 Časi odziva

Hitrost odziva aplikacije smo testirali na različnih iskalnih nizih. Odzivni časi so prikazani v tabeli 5.1. Iz tabele 5.1 vidimo, da traja prvo iskanje manj kot desetinko sekunde, razen v enem primeru, ko traja iskanje dve desetinki sekunde.

Trajanje iskanja 2 in iskanja 3 je v vseh primerih manjše od desetinke sekunde in manjše od trajanja prvega iskanja. Razlog za hitro delovanje polno-tekstovnega iskanja pripisujemo učinkovito implementiranemu polno-tekstovnem indeksu besed na bazi, ki smo ga seveda pred uporabo zgenerirali

s primernim ukazom. Na tem mestu je potrebno dodati, da bi razlike v trajanju različnih metod prišle do izraza šele pri večjih obremenitvah strežnika.

| **iskalni niz** | **iskanje 1 [ms]** | **iskanje 2 [ms]** | **iskanje 3 [ms]** |
|---|---|---|---|
| **pelod** | / | / | 7 |
| **peloda** | 25 | 4 | 3 |
| **multipla skleroza** | 60 | 7 | 6 |
| **poškodba porodne poti** | 221 | 33 | 62 |
| **spremenjenost zaradi stresa** | 33 | 28 | 21 |
| **trajna osebnostna spremenjenost po stresu** | 12 | 4 | 5 |

Tabela 5.1: Iskalni niz predstavlja diagnozo ali del diagnoze, po katerem iščemo ustrezen MKB. Različna iskanja so opisana v poglavju Rezultati. Iskanje 1 ustreza iskanju po lemah, iskanje 2 ustreza polno-tekstovnemu iskanju po MKB stolpcu opisov, iskanje 3 ustreza tekstovnemu iskanju po stolpcu opisov in po stolpcu lem.

## 5.2 Kvaliteta odgovorov

Kvaliteto izdelka običajno ocenjujemo glede na uporabno vrednost, ki jo izdelek prinaša. Izdelek je v največjo pomoč uporabniku takrat, ko v čim krajšem času postavi v ospredje tiste MKB-je, ki jih uporabnik išče. Pravilnost vrstnega reda izbora MKB-jev in odzivni čas smo uporabili za oceno kvalitete seznama MKB-jev, ki jih ta aplikacija vrne v odgovor na neko iskano frazo.

### 5.2.1 Človeška ocena

Bolj subjektivno pa smo primorani oceniti kvaliteto razporeditve MKB-jev, ki jih vrne aplikacija. V primeru iskalnih nizov ,,pelod“ in ,,peloda“ obstaja MKB z opisom ,,Alergijski rinitis zaradi peloda“. V primeru iskalnega niza ,,pelod“ iskanji 1 in 2 ostaneta brez zadetka. Pri iskanju 1 je razlog v delovanju lematizatorja, ki nepravilno lematizira besedo ,,pelod“ v lemo ,,peloda“ (pravilna lema je ,,pelod“). Leme ,,peloda“ v tabeli lem ni, saj se beseda med MKB-ji pojavi samo enkrat in to v obliki ,,peloda“, ki jo lematizator pravilno spremeni v lemo ,,pelod“. Iskanje 2 ne vrne zadetka, ker je beseda v opisu ustreznega MKB-ja v drugi obliki (,,peloda“), kot je v iskalni frazi (,,pelod“). Iskanje 3 vrne ustrezen zadetek, ker v stolpcu z lemami obstaja lema ,,pelod“.

Če kot iskalno frazo uporabimo ,,peloda“, vsi trije načini vrnejo ustrezen MKB kot prvi in edini zadetek.

Iskalna fraza ,,multipla skleroza“ nam v vseh treh primerih vrne kot prvi zadetek ustrezen MKB. Iskanje 1 nam vrne skupno sto MKB-jev, ostalih 99 vsebuje različne inačice besede multipla. Iskanje 2 nam vrne 13 zadetkov 3 so povezani z besedo ,,multipla“ in 11 z besedo ,,skleroza“. Iskanje 3 pa 15 zadetkov od katerih so 3 povezani z besedo ,,multipla“ in 13 z besedo ,,skleroza“ ali besedo ,,skleroze“.

Naslednja iskalna fraza je ,,poškodba porodne poti“. V tem primeru smo namenoma testirali frazo, ki nima ustreznega MKB-ja, ampak je uporabljena za zožitev iskanja. Iskanje 1 nam vrne 100 različnih MKB-jev. Številka 100 je prednastavljeni parameter za maksimalno število MKB-jev, ki nam jih aplikacija vrne. Od tega jih prvih 56 vsebuje besede kot so ,,porodna“ ali ,,porodne“, vsi MKB-ji pa vsebujejo še besede kot so ,,poškodba“ ali ,,poškodbe“.

Iskanje 2 nam prav tako vrne 100 zadetkov od katerih jih 61 vsebuje besede ,,porodna“, ,,porodom“, ,,porodne“ in ,,poškodba“, ,,poškodbe“. Za razliko od iskanja 1 se teh najboljših 61 MKB-jev ne pojavi na začetku seznama, kar pa ni nepremostljiva ovira, saj ima aplikacija IMKB filtrirno polje,

v katerega lahko vpišemo iskalno frazo, hkrati pa se bodo MKB-ji dinamično filtrirali glede na to, ali slovenski naziv, kategorija ali podkategorija vsebujejo iskano frazo.

Pri iskanju 3 so številke enake, kot pri iskanju 2, s to razliko, da so najbolj relevantni MKB-ji na začetku seznama.

Pri iskalnih frazah ,,spremenjenost zaradi stresa“ in ,,trajna osebnostna spremenjenost po stresu“ med načini iskanja izstopata 1 in 3, ki sta si pri kvaliteti razporeditve ustreznih MKB-jev podobna, pri čemer je način 3 hitrejši od načina 1. Zato je način 3 tudi priporočen način iskanja po MKB klasifikacijah.

Preizkusili smo še iskanje 4. Od iskanja 2 se razlikuje tako, da izvajamo polno-tekstovno iskanje po stolpcu angleških opisov. Trajanje tega iskanja je v vseh primerih manjše od desetinke sekunde. Za iskalne nize smo vzeli angleške prevode iskalnih nizov, ki smo jih uporabili za oceno delovanja drugih iskanj. Iskalni niz ,,pollen“ nam je vrnil dva zadetka, medtem ko je slovenska inačica ,,pelod“ vrnila samo en zadetek. Vzrok tiči v tem, da je v slovenskem jeziku v opisu enkrat beseda ,,peloda“ in enkrat ,,cvetni prah“. Pri iskalnem nizu ,,multiple sclerosis“ je prvih 6 MKB-jev enakih kot pri iskalnem nizu ,,multipla skleroza“ v iskanju izvedenem na način 3. Pri naslednjih zadetkih pa prihaja do manjših razlik.

Iskalno frazo ,,poškodbe porodne poti“ smo zamenjali z iskalno frazo ,,birth laceration“. Prva dva MKB-ja, ki jih vrne iskanje po angleških opisih, sta enaka kot pri slovenskem iskanju na način 3, potem pa prihaja do nekaterih razlik. Iskalno frazo ,,spremenjenost zaradi stresa“ smo zamenjali z iskalno frazo ,,change due to stress“. Iskanja sicer prineseta nekaj razlik, vendar pa so v obeh primerih diagnoze iskane kategorije (F62) na vrhu seznama vrnjenih MKB-jev. Iskalno frazo ,,trajna osebnostna spremenjenost po stresu“ smo zamenjali z iskalno frazo ,,enduring personality change after stress“ prvi štirje zadetki so enaki, med naslednjimi zadetki pa je tudi nekaj razlik.

### 5.2.2 Strojna ocena

Kvaliteto dobljenih rezultatov smo preizkusili tudi na 1000 MKB-jih, ki imajo vsaj dve besedi dolgi vsaj pet črk, kot iskalni vnos pa smo vzeli prvi dve besedi. Zanimalo nas je, kolikokrat se bo iskani MKB pojavil prvi na seznamu, kolikokrat med prvimi tremi, kolikokrat med prvimi desetimi in kolikokrat med prvimi stotimi prikazanimi MKB-ji (tabela 5.2.2). V brskalnik vračamo samo prvih sto odgovorov. V primeru, ko MKB-ja ni med prvimi 100 zadetki, pomeni, da smo ostali brez iskanega MKB-ja.

| | **top 1** | **top 3** | **top 10** | **top 100** |
|---|---|---|---|---|
| **iskanje 1** | 508 | 700 | 861 | 989 |
| **iskanje 2** | 567 | 759 | 899 | 999 |
| **iskanje 3** | 568 | 757 | 898 | 999 |

Tabela 5.2: Prikazano je število zadetkov iskane klasifikacije med vrhnjimi (top) rezultati. Top 1 predstavlja najvišje uvrščen rezultat, top 3 pa predstavlja primer, kjer se je iskana klasifikacija pojavila med tremi najvišje uvrščenimi rezultati. Različna iskanja so opisana v poglavju Rezultati. Iskanje 1 ustreza iskanju po lemah, iskanje 2 ustreza polno-tekstovnemu iskanju po MKB stolpcu opisov, iskanje 3 ustreza polno-tekstovnemu iskanju po stolpcu opisov in po stolpcu lem.

Iskani MKB je v več kot polovici primerov najvišje na seznamu rezultatov, v več kot 70% primerov je med prvimi tremi rezultati in v skoraj 90% primerov je med prvimi desetimi rezultati.

Iskanji 2 in 3 samo v enem primeru med vrnjenimi rezultati nista vrnila iskanega, gre za MKB s kategorijo A02 in podkategorijo 8. Opis tega MKB-ja se glasi: ,,Druge opredeljene salmonelne infekcije“. Prvi dve besedi sta besedi ,,druge“ in ,,opredeljene“, ta kombinacija pa se nahaja v skoraj 400 opisih. Ker uporabniku vrnemo največ 100 opisov, tega opisa ni bilo med prvimi stotimi. Ta primer ne odraža realnega primera uporabe, saj bi v tem primeru uporabnik bolj verjetno iskal ,,salmonelne infekcije“ v tem primeru pa je iskani MKB obakrat na drugem mestu.

Iskanje 1 se obnese primerljivo, a vendar nekoliko slabše. Razlog morda tiči v tem, da je polno-tekstovno iskanje bolj prilagojeno strojni oceni, kot je iskanje po lemah – človeška ocena je dala nekoliko drugačno sliko. Drugi možen razlog je, da smo pri iskanju 1 predpostavili, da je ključnega pomena za razvrstitev besed njihova dolžina, frekvence določene besede v vseh MKB-jih pa nismo uporabili pri diskriminaciji med besedami, ki sestavljajo iskalno frazo. Dolžina besede in njena frekventnost sta, seveda, korelirani. Kljub temu pa se je izkazalo, da se kot kriterij pri določanju ujemanja iskalne fraze in MKB-ja nekoliko bolje obnese frekventnost besede.

### Statistična signifikantnost rezultatov

Na prvi pogled se rezultati zdijo zadovoljivi, kljub temu pa smo želeli dokazati, da nismo do teh rezultatov prišli naključno, zato smo izračunali še statistično signifikantnost rezultatov. Urejenostna spremenljivka, ki smo jo opazovali, je zaporedno mesto, na katerem se določeni MKB pojavi med rezultati, ki so vrnjeni v brskalnik. Zanimalo nas je zaporedno mesto, kjer se pojavi MKB pri treh iskanjih, kot je opisano pri strojni oceni. Za primerjavo pa smo iz podatkovne baze izbrali naključnih 100 MKB-jev s pomočjo SQL funkcije RAND(). Ker je vseh MKB-jev skoraj 19.000, v večini primerov med naključno izbranimi 100 MKB-ji ni iskanega MKB-ja – v takem primeru smo predpostavili, da je bil MKB zapisan na 101. mestu – tako da je dejanska statistična signifikantnost naših rezultatov večja od izračunane. Pri generiranju seznama smo za iskalno frazo uporabili najprej prvi dve besedi, potem pa še naključne tri besede iz opisa MKB-ja. Iz centralnega limitnega izreka sledi, da je vsaka vsota ali povprečje naključnih spremenljivk, če je število členov dovolj veliko, približno normalno porazdeljena. [39] Neodvisna spremenljivka pri našem iskanju je mesto na seznamu MKB-jev, kjer se pojavi iskani MKB – za določeno kombinacijo besed v iskalni frazi lahko obstaja eden ali mnogo MKB-jev, ki to kombinacijo vsebujejo. Tako lahko za oceno statistične signifikantnosti rezultatov uporabimo Studentovo statistiko t. Tudi v primeru, da neodvisna spremenljivka ni normalno porazdeljena, lahko uporabimo Stu-

dentovo statistiko t, ker imamo zadostno število podatkov v vzorcu, hkrati pa ne poznamo variance vzorca vnaprej.

Naša domneva je bila, da so rezultati, ki jih dobimo z našimi iskanji, statistično signifikantno boljši (enostranski test), kot če bi MKB-je izbrali naključno. Nasprotna hipoteza je bila, da ne moremo trditi, da so rezultati, ki jih dobimo z našimi iskanji, statistično signifikantno boljši kot če bi MKB-je izbrali naključno. V primeru, da za iskalno frazo uporabimo prvi dve besedi MKB-ja, dobimo pri primerjavi prvega iskanja z naključnim t-vrednost 208,6, drugega iskanja 445,3 in tretjega iskanja 435,9 (tabela 5.2.2). V primeru, da smo za iskalno frazo uporabili naključne tri besede MKB-ja, dobimo pri primerjavi prvega iskanja z naključnim t-vrednost 558,0, drugega iskanja 658,2 in tretjega iskanja 650,0. Glede na to, da je v vseh šestih primerih prostostna stopnja 1998 (dva vzorca s tisoč podatki), lahko iz ustrezne tabele [18] odčitamo rezultat, kjer je v vseh šestih primerih verjetnost, da drži nasprotna hipoteza, manjša od 0,001. Torej je trditev, da so naši rezultati boljši od naključnih, pri stopnji zaupanja 95%, statistično signifikantna.

| **Studentova statistika t** | **prvi dve besedi** | **naključne tri besede** |
|---|---|---|
| **iskanje 1 in naključno** | 208,6 | 558,0 |
| **iskanje 2 in naključno** | 445,3 | 658,2 |
| **iskanje 3 in naključno** | 435,9 | 650,0 |
| **iskanje 2 in iskanje 1** | 4,10 | 1,95 |
| **iskanje 3 in iskanje 1** | 3,76 | 1,56 |
| **iskanje 2 in iskanje 3** | 0,57 | 0,46 |

Tabela 5.3: V tabeli je prikazana vrednost Studentove statistike t za primerjave iskanj z naključnim izborom MKB-jev in za primerjave iskanj med sabo.

Dodatno nas je zanimalo, če obstaja statistično signifikantna razlika v rezultatih naših metod, če jih primerjamo med seboj. Naša hipoteza pri vsaki primerjavi je bila, da prvo primerjano iskanje vrne boljše rezultate kot

drugo primerjano iskanje (enostranski test). Pri generiranju seznama smo za iskalno frazo uporabili najprej prvi dve besedi, potem pa še naključne tri besede iz opisa MKB-ja. V primeru, da smo za iskalno frazo uporabili prvi dve besedi MKB-ja, nam primerjava iskanja dva z iskanjem ena da t-vrednost 4,10, iskanja tri z iskanjem ena 3,76 in iskanja dva z iskanjem tri 0,57. V primeru, da smo za iskalno frazo uporabili naključne tri besede MKB-ja, nam primerjava iskanja dva z iskanjem ena da t-vrednost 1,95, iskanja tri z iskanjem ena 1,56 in iskanja dva z iskanjem tri 0,46. Iz ustrezne tabele [18] odčitamo rezultat, prostostna stopnja je v vseh primerih 1998. Pri primerjavi iskanja dva in iskanja ena lahko trdimo, da je iskanje dva boljše od iskanja ena s stopnjo zaupanja 95% v primeru treh naključnih besed in s stopnjo zaupanja 99,9% v primeru prvih dveh besed. Pri primerjavi iskanja tri z iskanjem ena lahko trdimo, da je iskanje tri boljše od iskanja ena s stopnjo zaupanja 85% v primeru treh naključnih besed in s stopnjo zaupanja 99,9% v primeru prvih dveh besed. Pri primerjavi iskanj dva in tri ne moremo trditi, da je eno iskanje boljše od drugega, saj je v primeru, da smo za iskalno frazo uporabili tri naključne besede ali prvi dve besedi, stopnja zaupanja take trditve le 50%. Pri stopnji tveganja $\alpha = 0{,}05$ lahko trdimo, da je iskanje dva statistično signifikantno boljše od iskanja ena v obeh primerih. Pri enaki stopnji tveganja lahko trdimo, da je iskanje tri statistično signifikantno boljše od iskanja ena v primeru prvih dveh besed, v primeru naključnih treh besed pa tega ne moremo trditi. Pri primerjavi iskanja dva in iskanja tri pa v nobenem primeru ne moremo trditi, da je eno iskanje statistično signifikantno boljše od drugega.

# Poglavje 6

# Zaključek

V okviru diplomske naloge smo izdelali aplikacijo IMKB v obliki spletne rešitve, ki nam, ob podani diagnozi ali iskalni frazi, pomaga določiti ustrezno mednarodno klasifikacijo bolezni. Tako smo dokazali obe hipotezi (opisani v 1.2), ki smo ju postavili na samem začetku diplomskega dela. Pri implementaciji iskanj po Mednarodni klasifikaciji bolezni (opisani v 2) smo na strežniku uporabili poslovno verzijo Jave (opisano v 3.1), na odjemalcu ogrodje Angular (opisano v 3.2), pri podatkovni bazi smo uporabili sistem za upravljanje podatkovnih baz MySQL (opisano v 3.3), pri računalniški obdelavi naravnega jezika pa smo uporabili orodje LemmaGen (opisano v 3.4.4).

Implementirali smo tri načine iskanja (opisane v poglavju 4) po Mednarodni klasifikaciji bolezni. Pri prvem iskanju smo besede iz iskalne fraze lematizirali in jih iskali v tabeli lem, pri drugem iskanju smo po polno-tekstovnem indeksu opisov bolezni iskali v načinu naravnega jezika, pri tretjem iskanju pa smo po polno-tekstovnem indeksu opisov bolezni in stolpcu z lemami iskali v Boolovemu načinu. V primeru, ko imamo v iskalni frazi veliko besed, bi se verjetno najbolje obnesel drugi način, saj polno-tekstovno iskanje v načinu naravnega jezika daje večjo težo besedam, ki se v MKB-jih pojavljajo redkeje. V primeru, ko smo manj gotovi, da se beseda v iskalni frazi nahaja v enaki obliki tudi v opisu MKB-ja, pa bi se verjetno bolje obnesla načina 1 in 3. Končno merilo katero od teh iskanj je najbolj uporabno, je katero od

teh iskanj končnemu uporabniku najbolj pomaga najti ustrezni MKB, kar pa lahko določi le vsak uporabnik sam.

Diplomsko delo smo zaključili z ocenami delovanja aplikacije (opisano v 5.2). Aplikacijo IMKB smo vrednotili po času, ki je potreben za izvedbo iskanja (opisano v 5.1) in po mestu na seznamu vrnjenih MKB-jev, ki ga zaseda iskani MKB (opisano v 5.2.2). Na podlagi lastne ocene, avtomatizirane ocene in statistične signifikantnosti rezultatov sklepamo, da je spletna aplikacija, ki je rezultat tega dela, koristno orodje pri klasifikaciji bolezni. Dodatni razlog, da sklepamo, da bi aplikacija, ki je plod diplomskega dela, utegnila biti uporabna, je, da obstaja plačljivo orodje za iskanje po MKB-jih [19], ki pa ponuja le preprosto iskanje MKB-jev za določen niz in je po fleksibilnosti bistveno skromnejše od našega IMKB.

# Literatura


[1] Angular - JavaScript ogrodje. https://angular.io/. Dostopano dne: 18. januar 2017.

[2] AngularJS: Introduction. https://docs.angularjs.org/guide/introduction. Dostopano dne: 21. maj 2016.

[3] Avstralska modifikacija desete revizije mednarodne klasifikacije bolezni in sorodnih zdravstvenih problemov za statistične namene (MKB-10-AM). http://www.nijz.si/sl/podatki/mkb-10-am-verzija-6. Dostopano dne: 21. januar 2017.

[4] Hibernate - objektno relacijski preslikovalnik. http://hibernate.org/. Dostopano dne: 18. januar 2017.

[5] International Classification of Diseases. http://www.who.int/classifications/icd/en/. Dostopano dne: 19. november 2016.

[6] International Classification of Diseases (ICD) Information Sheet. http://www.who.int/classifications/icd/factsheet/en/. Dostopano dne: 6. februar 2017.

[7] International Statistical Classification of Diseases and Related Health Problems, Tenth Revision, Australian Modification. https://www.accd.net.au/Icd10.aspx. Dostopano dne: 6. december 2016.

[8] Introduction to natural language processing. http://www.londoninternational.ac.uk/sites/default/files/

computing-samples/co3354_ch1-3.pdf. Dostopano dne: 11. december 2016.

[9] Is it more conveniet to work directly with JPA or is it better to code directly with ORM implementation. http://programmers.stackexchange.com/questions/260409/is-staying-implementation-agnostic-really-worth-it. Dostopano dne: 21. marec 2016.

[10] Is knowledge of SQL necessary if you use object–relational mappers. http://programmers.stackexchange.com/questions/100939/is-sql-important-if-i-know-orm-frameworks-well. Dostopano dne: 20. marec 2016.

[11] Jackson Project. https://github.com/FasterXML/jackson. Dostopano dne: 21. januar 2017.

[12] Java Platform, Enterprise Edition, Tutorial. https://docs.oracle.com/javaee/7/tutorial/. Dostopano dne: 21. maj 2016.

[13] jQuery - JavaScript knjižnica. https://jquery.com/. Dostopano dne: 18. januar 2017.

[14] Lemmagen4J – Javanska implementacija orodja Lemmagen. https://github.com/szitnik/Lemmagen4J/. Dostopano dne: 5. november 2016.

[15] Mednarodna klasifikacija bolezni in sorodnih zdravstvenih problemov za statistične namene. http://www.nijz.si/sites/www.nijz.si/files/uploaded/podatki/klasifikacije_sifranti/mkb/mkb10-am-v6_v03_splet.pdf. Dostopano dne: 6. februar 2017.

[16] MySQL Connector/J 5.1. http://dev.mysql.com/doc/connector-j/5.1/en/. Dostopano dne: 7. december 2016.

[17] MySQL Internals - Full-Text Search. `https://dev.mysql.com/doc/internals/en/full-text-search.html`. Dostopano dne: 5. februar 2017.

[18] NIST/SEMATECH e-Handbook of Statistical Methods: Critical Values of the Student's t Distribution. `http://www.itl.nist.gov/div898/handbook/eda/section3/eda3672.htm`. Dostopano dne: 10. december 2016.

[19] Nova vizija d.d.: Program MKB-10. `http://www.vizija.si/medicinska-informatika/mkb/`. Dostopano dne: 5. januar 2017.

[20] Object–relational impedance mismatch. `https://en.wikipedia.org/wiki/Object-relational_impedance_mismatch`. Dostopano dne: 15. marec 2016.

[21] React - JavaScript ogrodje. `https://github.com/reactjs`. Dostopano dne: 18. januar 2017.

[22] RESTEasy - frameworks that help you build RESTful Web Services and RESTful Java applications. `http://resteasy.jboss.org/`. Dostopano dne: 21. januar 2017.

[23] Tf-idf (term frequency–inverse document frequency). `https://en.wikipedia.org/wiki/Tf-idf`. Dostopano dne: 5. februar 2017.

[24] Undertow - javanski spletni strežnik. `http://undertow.io/`. Dostopano dne: 18. januar 2017.

[25] Wikipedija: Georgetown-IBM experiment. `https://en.wikipedia.org/wiki/Georgetown-IBM_experiment`. Dostopano dne: 21. maj 2016.

[26] Wikipedija: Lematizacija. `https://sl.wikipedia.org/wiki/Lematizacija`. Dostopano dne: 11. december 2016.

[27] Wikipedija: MySQL. https://en.wikipedia.org/wiki/MySQL. Dostopano dne: 21. maj 2016.

[28] Wikipedija: NLP. https://en.wikipedia.org/wiki/Natural_language_processing. Dostopano dne: 21. maj 2016.

[29] Wikipedija: object–relational mapping. https://en.wikipedia.org/wiki/Object-relational_mapping. Dostopano dne: 15. marec 2016.

[30] Wikipedija: WildFly. https://en.wikipedia.org/wiki/WildFly. Dostopano dne: 21. maj 2016.

[31] Wildfly - javanski aplikacijski strežnik. http://wildfly.org/. Dostopano dne: 18. januar 2017.

[32] Christian Bauer and Gavin King. *Java Persistance with Hibernate*. Dreamtech Press, 2006.

[33] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[34] Chris J Date. *SQL and relational theory: how to write accurate SQL code*. "O'Reilly Media, Inc.", 2011.

[35] Russell J.T. Dyer. *MySQL in a Nutshell*. "O'Reilly Media, Inc.", 2008.

[36] izr. prof. dr. Matjaž Kukar. *Gradivo predmeta: Tehnologija upravljanja podatkov*. "Fakulteta za računalništvo in informatiko, Univerza v Ljubljani", 2015/2016.

[37] Ivar Jacobson. *Object-oriented software engineering: a use case driven approach*. Pearson Education India, 1993.

[38] Matjaž Juršič, Igor Mozetič, Tomaž Erjavec, and Nada Lavrač. Lemmagen: Multilingual lemmatisation with induced ripple-down rules. *Journal of Universal Computer Science*, 16(9):1190–1214, 2010.

[39] prof. dr. Aleksandar Jurišić. *Gradivo predmeta: Verjetnost in statistika.* "Fakulteta za računalništvo in informatiko, Univerza v Ljubljani", 2013/2014.

[40] prof. dr. Matjaž Branko Jurič. *Gradivo predmeta: Postopki razvoja programske opreme.* "Fakulteta za računalništvo in informatiko, Univerza v Ljubljani", 2014/2015.

[41] Baron Schwartz, Peter Zaitsev, and Vadim Tkachenko. *High performance MySQL: Optimization, backups, and replication.* "O'Reilly Media, Inc.", 2012.

[42] Filippe Costa Spolti. *WildFly: New Features.* "Packt Publishing Ltd.", 2014.